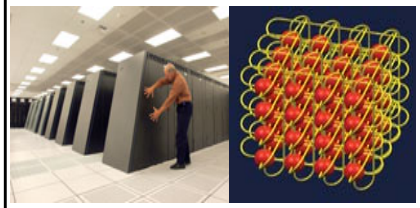
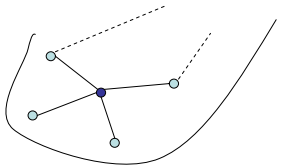
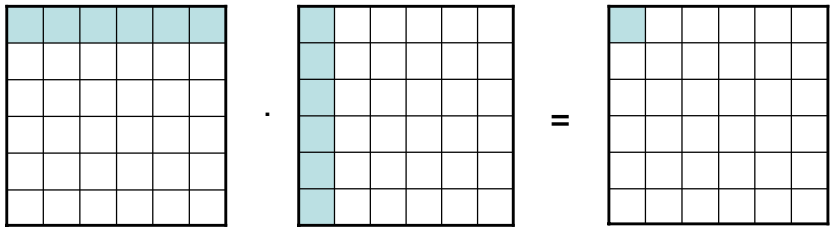
INF 3130/4130

1. Strengsøk – søk etter generelle patterns.
2. Dynamisk programmering – generell metode, lagdelt/trestruktur.
3. Flyt/Matching – egnet for løsning av “2-problemer”.
4. Søketrær – AVL, Splay bedre enn binære trær.
5. Prioritetskøer – det er ofte kø.
6. Søk (DFS, BFS, A*) – søk i tilstandsrom, Kunnskapsbaserte systemer.
7. Kompleksitetsteori.
    1. Uavgjørbarhet – ingen algoritmer.
    2. NP-kompletthet – tilnærming, probabilistiske metoder.
    3. P.
8. Parallelle og distribuerte algoritmer.

Parallelle og distribuerte algoritmer

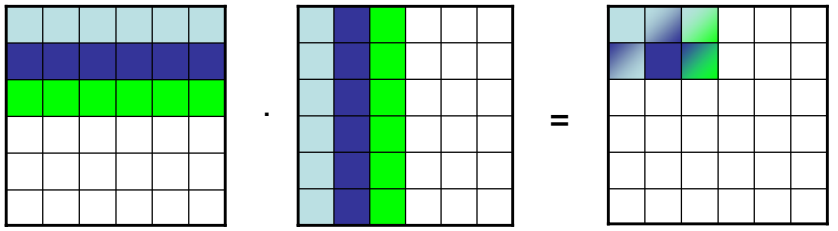
Parallelle datamaskiner Blue Gene/L, Cray Red Storm, BGW, ASC Purple	**Distribuerte nettverk**
• Topologien kjent. • Alle kan kommunisere med alle, ”direkte”. Blue Gene/L (IBM) (Lawrence Livermore) 131.072 prosessorer 367 TFlops (367 billioner flyttallsoperasjoner/sekund)  Top 500 supercomputer sites	• Topologien ikke kjent – en prosessor kjenner bare naboene sine. • En melding kan bare sendes til en nabo.  LAN, Internet, mobile ad-hoc nett, trådløse nett, etc.

Parallell matrisemultiplikasjon



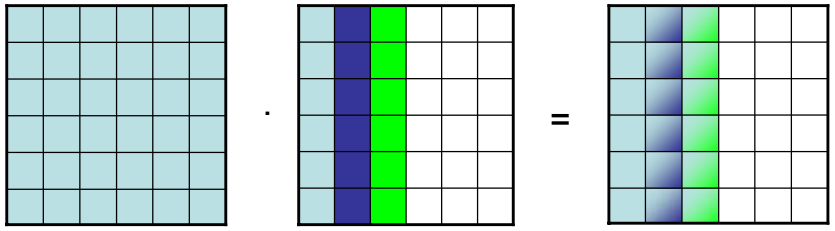
A B C = AB

Parallell matrisemultiplikasjon



A B C = AB

Parallell matrisemultiplikasjon



A B C = AB

Parallellprogrammering

MPI (Message Passing Interface) brukes ofte på store parallelle maskiner.

- Bibliotek (C og Fortran)
- Lar oss bygge et logisk nettverk av prosessorer
- Lar oss sende data mellom prosessorer

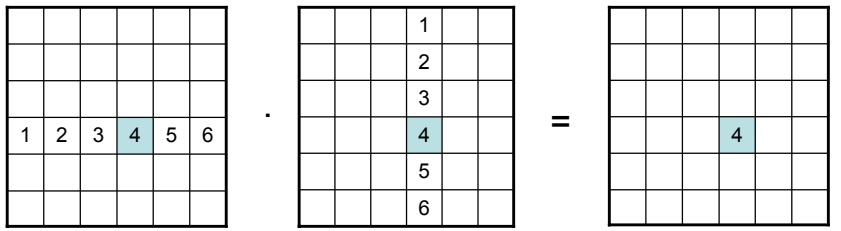
Parallellprogrammering

MPI (Message Passing Interface) brukes ofte på store parallelle maskiner.

- Bibliotek (C og Fortran)
- Lar oss bygge et logisk nettverk av prosessorer
- Lar oss sende meldinger mellom prosessorer og synkronisere mellom steg

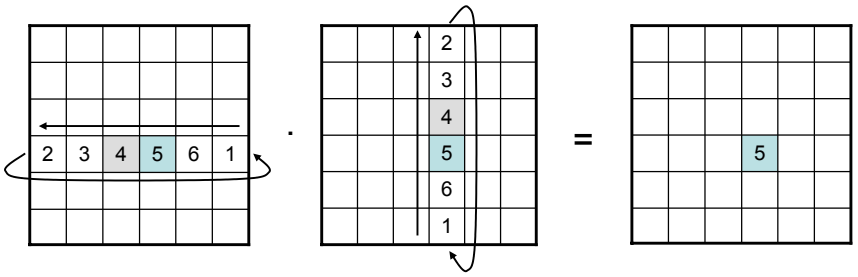
```
<Les inn data>                          // En prosessor har ansvaret for
<Fordel passende data til prosessorene> // å lese inn og kringkaste data
<Synkroniser>                           // Vent til alle har fått data
WHILE <ikke ferdig>
   <Gjør beregning med dataene>         // Gjøres av hver prosessor
   <Synkroniser>                        // Vent til alle er ferdig
   kommuniser med andre prosessorer     // Prosessorene får nye data
ENDWHILE
<Samle sammen data fra prosessorene>    // En prossor har ansvaret for
<Skriv ut resultat>                     // å samle sammen delsvar og gi
                                        // svar på beregningen
```

Parallell matrisemultiplikasjon (Cannon)



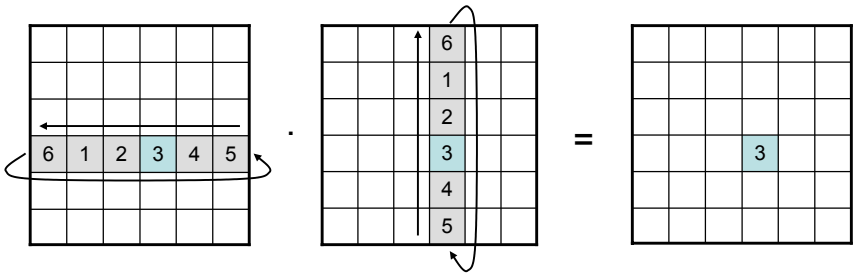
4*4

Parallell matrisemultiplikasjon (Cannon)



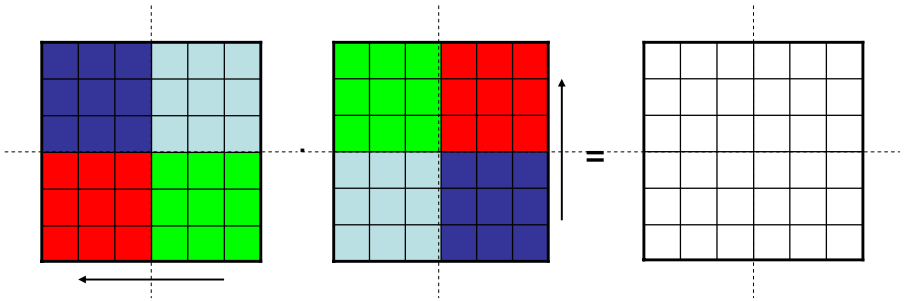
4*4
+5*5

Parallell matrisemultiplikasjon (Cannon)



4*4
+5*5
+6*6
+1*1
+2*2
+3*3

Parallell matrisemultiplikasjon (Cannon)

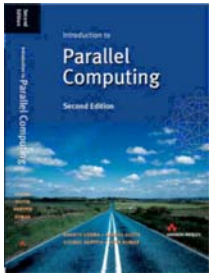


Parallellprogrammering

Message Passing Interface (MPI) er en kommunikasjonsprotokoll. De facto kommunikasjonsstandard for maskiner med distribuert minne.

OpenMP er et bibliotek som støtter multiprosessering på maskiner med delt minne.

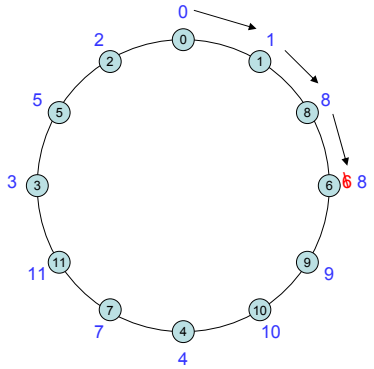
Introduction to Parallel Computing (2.ed). Grama, Karypis, Kumar, Gupta.



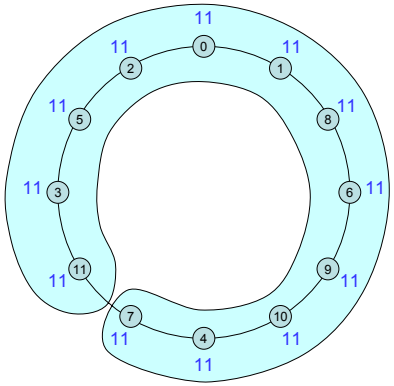
Algoritmer i distribuerte nettverk

- Topologien ikke kjent:
  - kjenner ikke strukturen,
  - kjenner ikke antall prosessorer.
- Ingen sentralisert kontroll.
- Som regel grafproblemer vi ser på:
  - nettverket er grafen (nettverket er inndata for algoritmen).
- Trenger rutiner for å legge struktur på nettet vårt.
- Algoritmene går i steg (synkron modell):
  - alle prosessorene gjennomfører en blokk med kode, før de går videre.
- Tidskompleksitet (antall steg).
- Kommunikasjonskompleksitet (antall meldinger).

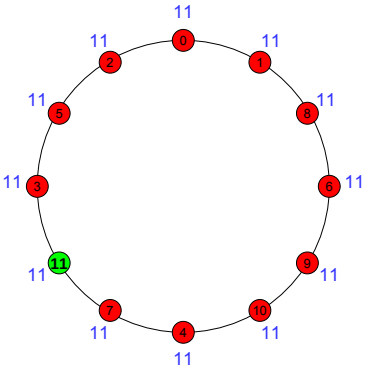
Ledervalg



Ledervalg



Ledervalg



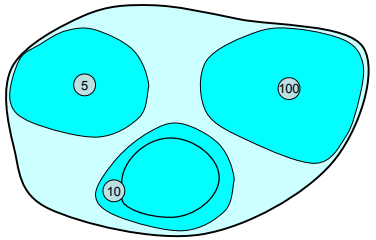
```
proc RingLeaderElection
    forall v do
        v.maxUID = v.UID
        send v.maxUID to clockwise neighbour

        while not < all v have received "leder found" message > do
            if u.maxUID received from counterclockwise neighbour of v then
                if u.maxUID < v.maxUID then
                    send v.maxUID to clockwise neighbour
                endif
                if u.maxUID > v.maxUID then
                    v.maxUID = u.maxUID
                    send v.maxUID to clockwise neighbour
                endif
                if u.maxUID = v.UID then
                    v.leaderStatus = "leader"
                    send "leader found" to clockwise neighbour
                endif
            endif

            if "leader found" received from counterclockwise neighbour of v then
                if v.leaderStatus ≠ "leader" then
                    v.leaderStatus = "not leader"
                    send "leader found" to clockvise neighbour
                endif
            endif
        endwhile
    endforall
endproc
```

Tidskompleksitet : $O(n)$
Meldingskompleksitet : $O(n^2)$

Algoritmer i distribuerte nettverk

I generelle nettverk er det litt mer komplisert enn i ringer…

- Vi kan ikke vite om en node som får tilbake sin egen UID er leder, eller om denne UIDen bare har gått en "liten runde".



- Kjenner vi diameteren d i nettverket, vet vi at vi må vente d steg før vi kan utpeke en leder. (Diameteren til en graf er lengden av lengste korteste sti mellom noe par av noder.)

Ledervalg

```
proc LeaderElection
    forall v do
        v.maxUID = v.UID
    endforall

    for round = 1 to d do
        forall v do
            send v.maxUID to all neighbours in N(v)
            for < each u.maxUID received > do
                if v.maxUID < u.maxUID then
                    v.maxUID = u.maxUID
                endif
            endfor
        endforall
    endfor
    forall v do
        if v.maxUID = v.UID then
            v.leaderStatus = "leader"
        else
            v.leaderStatus = "not leader"
        endif
    endforall
endproc
```

Tidskompleksitet : $O(d)$
Meldingskompleksitet : $O(dm)$

Kringkasting og BFS

- Kringkasting av informasjon, og innsamling av informasjon er fundamentale operasjoner innen distribuert og parallell databehandling.

- En node får f.eks ansvar for å les inn data fra en fil, så fordele arbeid rundt på ulike prosessorer, og til slutt samle inn delsvar og gi et svar.

Kringkasting og BFS

Flooding

1. En node som skal kringkaste sender sin melding til alle sine naboer.
2. Alle noder som mottok meldingen, sender denne videre til sine naboer.
3. Som sender den videre til sine naboer.
4. …

- Brukes i dynamiske nettverk.
- Ulempen er at det blir mye kommunikasjon.
- Trenger en metode for å avgjøre når vi er ferdige (f.eks diameter).

Kringkasting og BFS

Kringkasting i trær

I et tre kan vi implementere kringkasting effektivt:

- Roten starter, meldingen sendes nedover i treet, helt til den når bladnodene.
- Hver node må ha en liste over sine barn.

Kringkasting og BFS

Kringkasting i trær

Kjenner vi ikke strukturen i treet (foreldre / barn), kun roten, må vi bygge opp trestrukturen.

1. Bladnodene (unntatt en eventuell rot) ser at de bare har *en* nabo *v*, og sier ifra til *v* at de er barn av denne.
2. Når en internnode (unntatt en eventuell rot) har mottatt barnemeldinger fra alle unntatt *en* nabo *w,* sier den fra til *w* at den er barn av denne.
3. Vi fortsetter helt til roten har fått barnemeldinger fra alle sine naboer.

Kringkasting og BFS

Kringkasting i trær (convergecast)

Beregninger kan nå gjøres bottom-up i treet:

1. Bladnodene sender sin verdi (f.eks en sum) til forelderen.
2. Når en node har fått verdier fra alle sine barn gjøres en beregning, og verdien sendes videre til dennes forelder.

Kringkasting og BFS

Bredde først-trær

Vi har jo som regel ikke tre, men ønsker å legge en trestruktur oppå den generelle grafen (nettverket) vi har – et spenntre.

1. Initielt er treet tomt. Alle noder har en **`inTree`**-variabel som er FALSE.
2. Roten (utpekt på forhånd) setter **`inTree`** til TRUE, og sender ”join tree” til alle sine naboer.
3. Noder som mottar ”join tree”, velger ut *en* node den mottok meldingen fra (de kan være flere), og sender ”your child” til denne, og ”not your child” til de andre den mottok ”join tree” fra. **`inTree`** setter til TRUE, forelderen huskes.
4. I neste runde sender noder som akkurat ble med i treet ”join tree” til de av sine barn som enda ikke er med i treet. Er alle naboer med i treet, sendes ”terminated” til forelderen. Er ”terminated” mottatt fra alle barn, sendes også ”terminated” til forelderen.
5. 3 og 4 gjentas til roten mottar ”terminated” fra alle sine barn.

Kringkasting og BFS

```
proc BFSDistributed(s)
    s.parent = NIL
    s.inTree = TRUE
    s sends ”join tree” to all neighbours

    forall v do
        if v.inTree = FALSE and ”join tree” received then
            < choose a node u that has sent ”join tree” >
            v.parent = u
            v.inTree = TRUE
            send ”your child” to u
            send ”not your child” to (N(v) - u)
            send ”join tree” to N(v)
        else
            if child status message is received from all neighbours then
                if < v has no children > then
                    send ”terminated” to v.parent                    // v er bladnode
                else
                    if < ”terminated” received from all children > then
                        send ”terminated” to v.parent
                        halt                                         // v stopper
                    endif
                endif
            endif
        endif
    endforall
endproc
```

Ledervalg – igjen

Bredde først-trær

Tidligere brukte vi diameter-trikset for å finne en leder i en generell graf. Vi kan bruke spenntrær.

1. For hver node *s* i grafen, beregner vi et spenntre med *s* som rot. (Ett tre for hver node, beregnes uavhengig av hverandre.)
2. Hver node *s* gjøres så til mål for en convergecast av maxUID i treet den er rot i.
3. Noden som ender opp med maxUID lik sin egen UID er leder.

Korteste stier

Bredde først-trær

Et BFS-tre T_s med rot i *s* er et korteste sti-tre (færrest antall kanter) mellom *s* og de øvrige nodene i grafen.

Vi kan også bruke en distribuert variant av Bellmann-Ford-algoritmen for å finne korteste stier når kantene har ulike vekter.

Korteste stier (Bellmann-Ford)

```
proc BellmannFordDistributed(s)
    forall v do
        v.dist = ∞
        v.parent = ∞
    endforall

    s.dist = 0
    s.parent = 0

    for round = 1 to n – 1                                  // n antaes kjent
        forall v do
            send v.dist to all nodes in N_out(v)

            < after receiving u.dist from all u in N_in(v) >
            if v.dist > u.dist + w(uv) for u ∈ N_in(v)  then
                v.parent = < u with lowest u.dist + w(uv) >
                v.dist = min {v.dist, u.dist + w(uv) for u ∈ N_in(v)}
            endif
        endforall
    endfor
endproc
```

Tidskompleksitet : $O(n)$
Meldingskompleksitet : $O(nm)$

Korteste stier (Floyd)

Korteste stier alle til alle.

Floyds algoritme går i n steg. Nodene har numre 0 til n -1, og i k'te steg gir vi oss lov til å bruke node k som internnode i stien vi bygger opp mellom to noder i og j.

I den distribuerte varianten antar vi at det finnes en total ordning (<) av nodene, f.eks ved hjelp av UID. (Prosessorer i et datanett har alltid en slags ID, f.eks en IP-adresse. Helt anonyme nett mer en slags matematisk finurlighet.)

Korteste stier (Floyd)

```
proc FloydDistributed(s)
    forall v do
        forall u do
            if u ∈ N_out then
                v.distTable[u] = w(vu)
                v.firstTable[u] = u
            else
                if u = v then
                    v.distTable[u] = 0
                else
                    v.distTable[u] = ∞
                endif
            endif
        endforall
    endforall

    for pass = 0 to n - 1 do
        < let p be the next node in the node ordering >
        < p broadcasts p.distTable to all other nodes >
        forall v do
            forall u do
                if  v.distTable[u] > v.distTable[p] + p.distTable[u] then      // kortere fra v til u via p
                    v.distTable[u] = v.distTable[p]  + p.distTable[u]
                    v.firstTable[u] = v.firstTable[p]
                endif
            endforall
        endforall
    endfor
endproc
```

Asynkron modell

- Vi har hittil antatt at algoritmene våre går stegvis. Innenfor hvert steg har det ikke vært så nøye hvor fort ting gjøres, så lenge vi venter til alle er ferdige med å gå til neste steg.
- Dette er kanskje ikke den mest nøyaktige modellen av Den Virkelige Verden.
- Vi lar kommunikasjonskanalene i nettverket (kantene i grafen) ha buffere for meldinger.