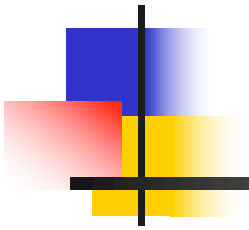


WHAT?? CAN'T YOU EVEN DO SIMPLE PLANNING??

- HEY, what just happened here??
- A sketch showing problems with synchronization, see the video.

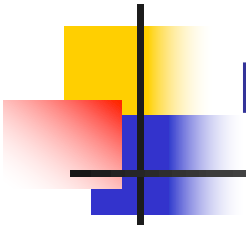
- Can YOU identify ALL the synchronization errors, issues, and problems??

- LESSON: Synchronization is HARD to get right!



IN3030 L4, v2022 – About synchronization, Moore's Law, the speed of light, and caches

Eric Jul
Programming Technology Group
Department of Informatics
University of Oslo

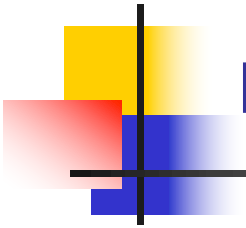


Review uke 3

1. Oblig plan
2. Amdahl og Gustafsons love
3. Modell(er) for hvordan vi programmerer
4. Viktige regler om lesing og skriving på felles data.
5. Synlighetsproblemet – «memory is not memory»
    1. Hvilke verdier ser ulike tråder som leser variable som en annen tråd skriver på?
6. Hvor fort kan JIT-kompilert kode gå?
7. Prefetch-mekanismen i elektronikken
8. Oblig 1: finn k største
9. Java har 'as-if sequential' semantikk
10. Effekten på eksekveringstid av cache

I. Moore's Law

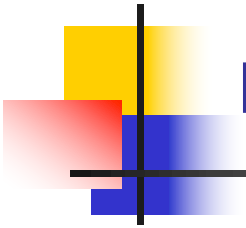
II. Speed of light



Plan for uke 4

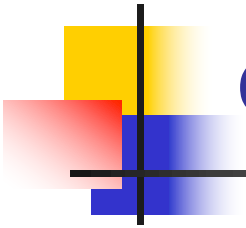
- Synchronization is HARD and ERROR PRONE!

I. Moore's Law
II. Speed of light
III. Why distribution
IV. Caches: WHAT and WHY
V. Matrice-multiplikasjon
VI. How to present your timing results
VII. Hva er PRAM modellen - og hvorfor er den ubrukelig for oss
VIII. Matrice-multiplikasjon



Moore's Law

- Used to be known as: «CPU speeds double every 2 years»



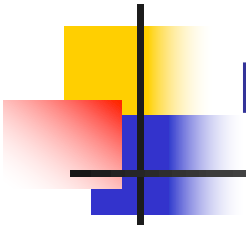
Original Moore's Law

- Transistors per square cm doubles:
- 1965 Article: Every year
- 1975 Article: Every two years

Perspektiv: Utvikling i CPU, minne, nettverk og mere om Moore's law

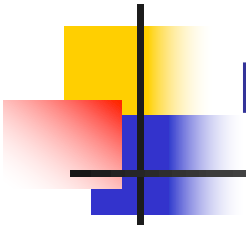
- 1980 CPU: Intel 8080 8-bit processor
  - «int»: one byte
  - «long»: two bytes
  - CPU clock frequency: 1 MHz
  - Memory: 64 kilobytes max, *i.e.*, 16 bit addresses (2 bytes)
  - 128 kB floppy disk
  - Data transmission 1.2 kbit/s == 150 bytes/s

- A look at Moore's law
  - Moore's original prediction 1965
  - Moore's revisted prediction 1975



Moore's Law Summary of Effects

- 1958-1975
  - Doubling of transistors every year
  - Derived effect: doubling of CPU speeds
- 1975-2005
  - Doubling of transistors every two years
  - Derived effect: doubling of CPU speeds – screaching halt at about 3 GHz
- 2005-2020
  - Doubling of transistors every two years
  - Derived effect: doubling of number of CORES *or* doubling of the amount of on-chip cache



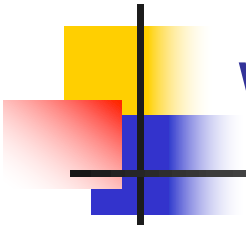
Newer Machines

- 2020: My Macbook Pro w Intel i7
  - 2.4 up-to 3.2 GHz – 8x cores
  - 16 Gbyte Memory
  - 10 Gbit/s network
  - 1 GB Solid State Disk (SSD)

Performance 1980 vs 2018

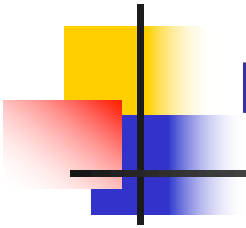
- 2018: Intel i7
  - 1 MHz CPU vs 8 x 3.2 GHz ~ factor 25,600
  - 16 Gbyte Memory vs. 64 kbytes ~ factor 250,000
  - 10 Gbit/s network vs 1 kbit/s ~ factor 10,000,000
  - 128 kB floppy disk vs 1 TB SSD ~ factor 8,000,000
  - 1200 baud serial line vs 1 Gbit/s Ethernet ~ factor 800,000

- ☺



What about ping time?

- Ping: sending an empty message from Oslo to Seattle & back
- How long does this take 1988 vs 2022?

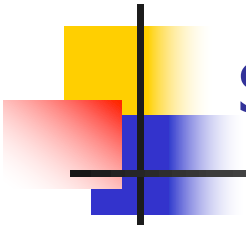


Round-trip ping Time

- 1988: Approximately 180-200 ms
- 2022: How much faster?
  - 1,000,000x faster?
  - 100,000x faster?
  - 10,000x faster?
  - 1,000x faster?
  - 100x faster?
  - 10x faster?
  - Same time?

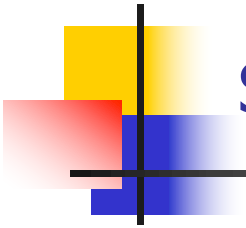
Latency has ***not*** changed: dominated and limited by the *incredibly* slow speed of light!

- How fast is the speed of light approximately?
- Great circle route over and back 16,000 km
- Speed of light in fiber 11/15, copper 3/5
- Great Circle round trip time: minimum of about 100 ms
- So ping time will ***NEVER*** go below 100 ms!



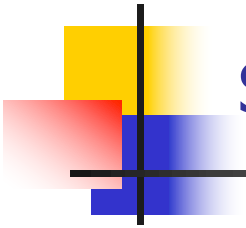
Speed of light revisited

- How fast is the speed of light approximately?
  - 300,000 km/s
  - Exact value?
- Speed of light
  - in fiber 11/15: about 220,000 km/s
  - in copper 3/5: about 180,000 km/s
- How far does light travel in 1 ns?
- When a 3 GHz core executes one cycle, how far does light travel?



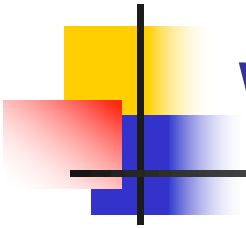
Speed of light revisited yet again

- How fast is the speed of light approximately?
  - 300,000 km/s
- How far does light travel in 1 ns?
- How far in copper in 1 ns?
- When a 3 GHz core executes one cycle, how far does light travel?



Speed of light: answers to questions

- How fast is the speed of light?
  - About 300,000 km/s
  - 299,792,458 m/s EXACTLY – by definition
- How far does light travel in 1 ns?
  - About 30 cm – call it a *lightfoot*
- How far in copper in 1 ns?
  - About 18 cm
- When a 3 GHz core executes one cycle, how far does light travel in vacuum and in copper?
  - About 10 cm (vacuum) and 6 cm (copper)



WHAT is a cache?

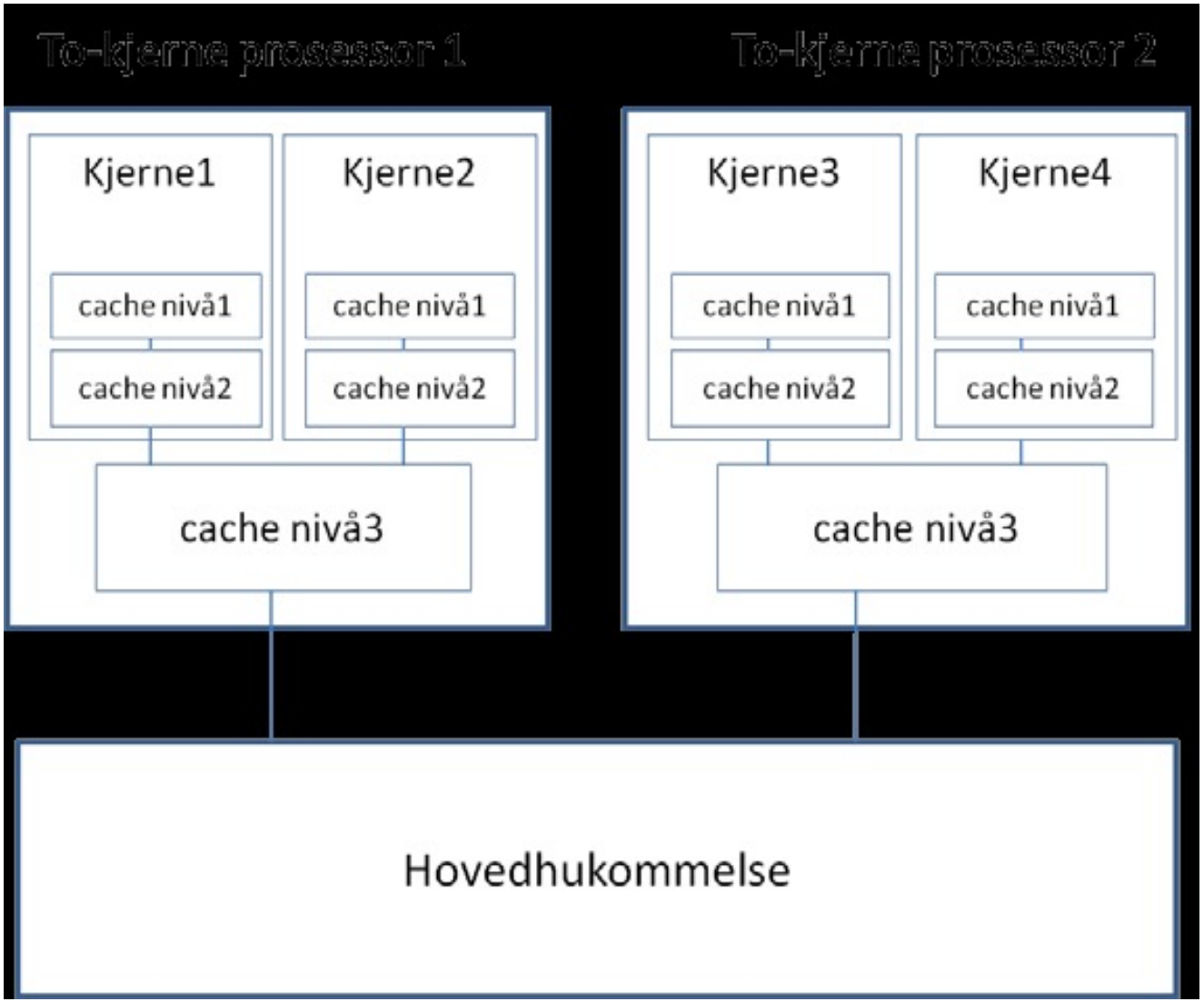
- Principle of a cache
- Speed of memory vs size
- Second reason: Speed light

Maskin 1980 (uten cache)

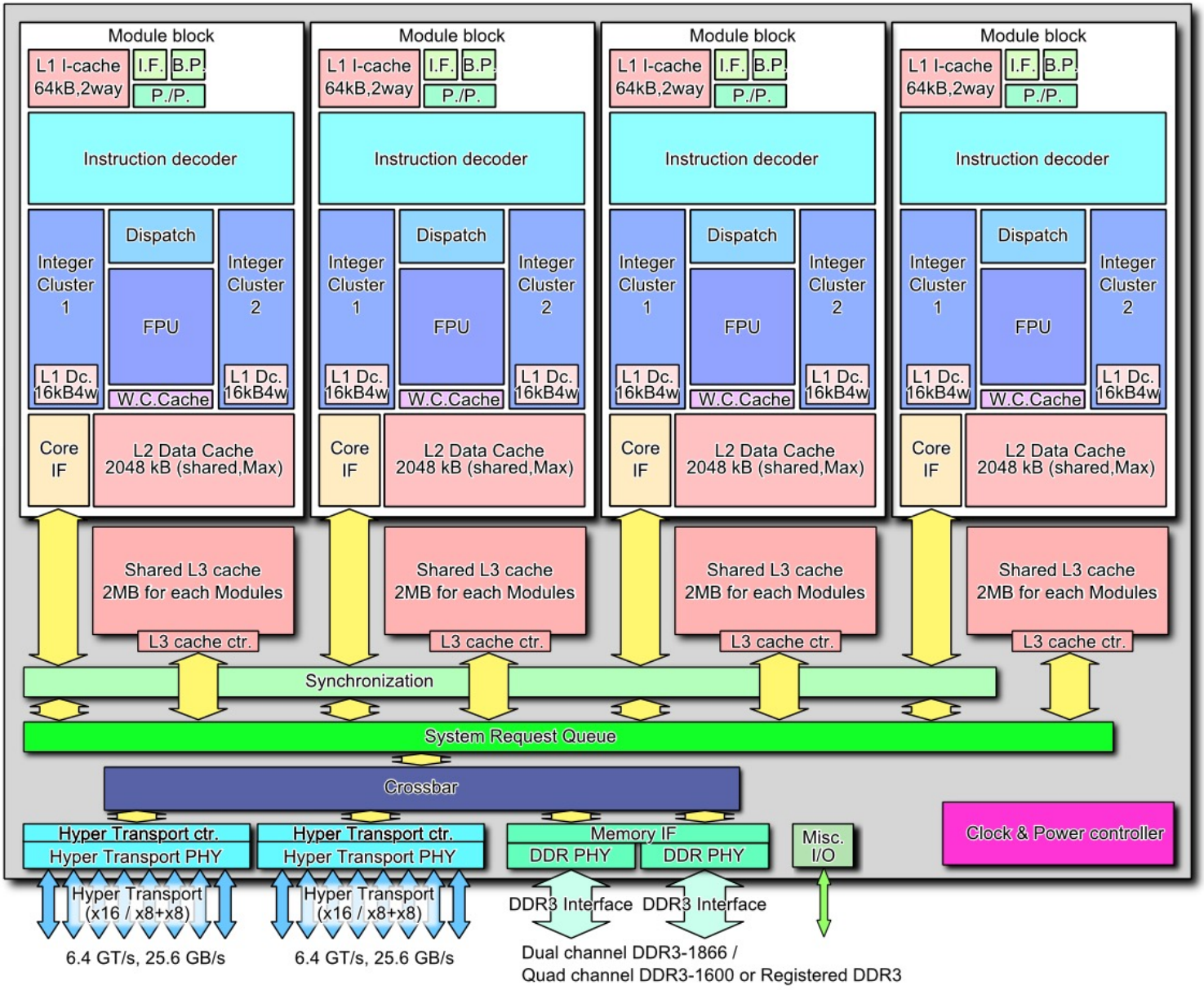


Figur 19.1 Skisse av en datamaskin i ca. 1980 hvor det bare var én beregningsenhet, en CPU, som leste sine instruksjoner og både skrev og leste data (variable) direkte i hovedhukommelsen. Intel 8080: 1 MHz CPU

Maskin ca. 2010 med to dobbeltkjerne CPU-er



Hukommelses-systemet i en 4 kjerne CPU – mange lag og flere ulike beregningsmoduler i hver kjerne.:

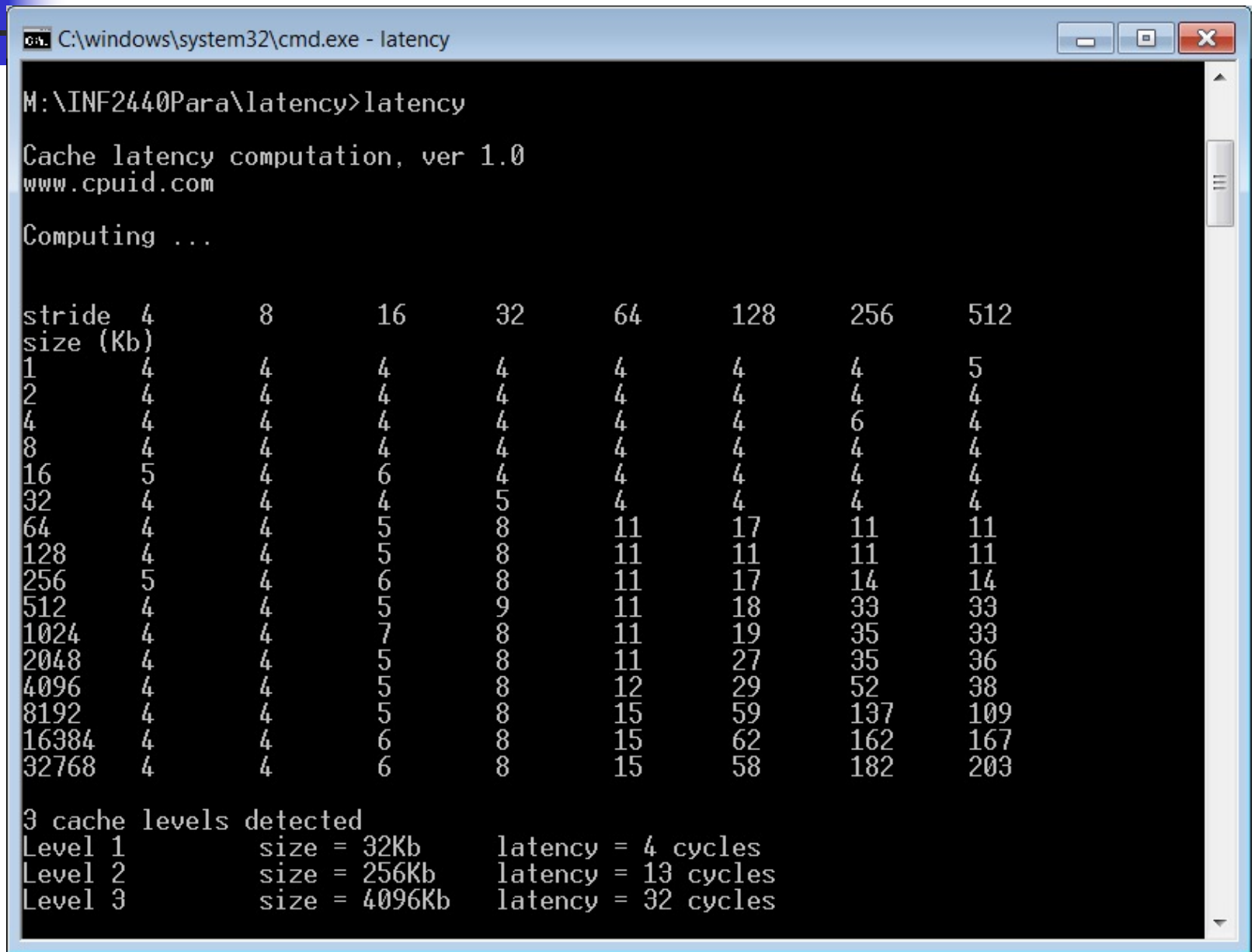


Hvordan tar vi hensyn til cache-systemet for å få raskere programmer?

- Vi ser bare på data-cachene (lite å hente på instruksjonene)
- Viktig å vite er at hver gang vi skal hente data i hovedlageret , får vi en cach-linje = 64 byte = f.eks 8 heltall (int)
- Det er svært begrenset plass i cachene, og en cach-linje som ikke har vært brukt på 'lenge' vil bli 'kastet ut'(overskrevet av en annen, nyere) cache-linje
- Slik er raskest:
  - Jobber på få data (korte deler av en array) 'lenge' av gangen – ikke hoppe rundt.
  - Helst gå forlengs eller baklengs gjennom data (arrayene) (i, i+1,.. eller: i, i-1,..)

Vi må lage slike cache-vennlige programmer !

Test av forsinkelse i data-cachene og hovedhukommelsen - latency.exe (fra CPUZ)

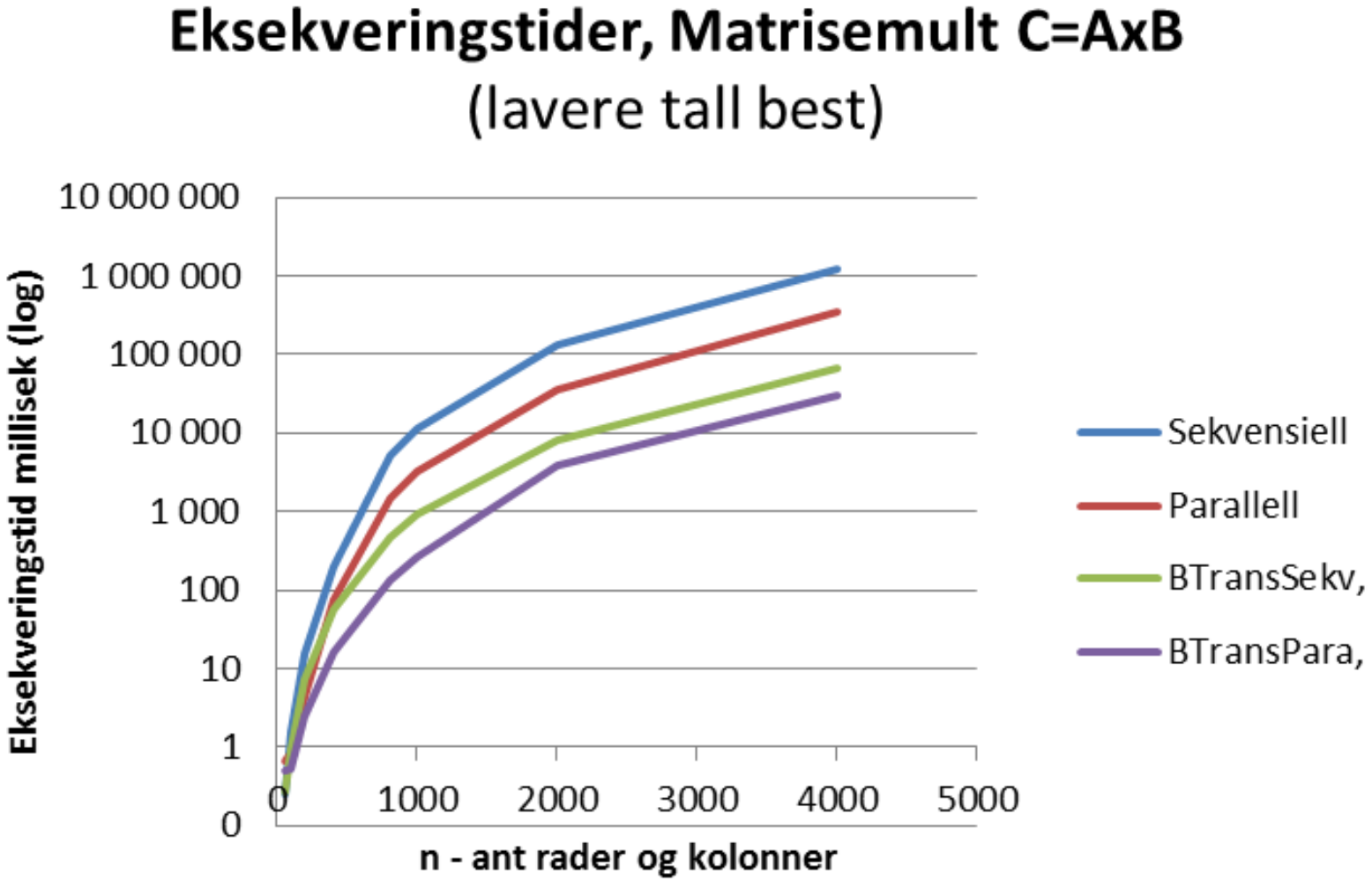


Oppsummering – ideen om at vi har *uniform* aksesstid i hukommelsen er helt galt

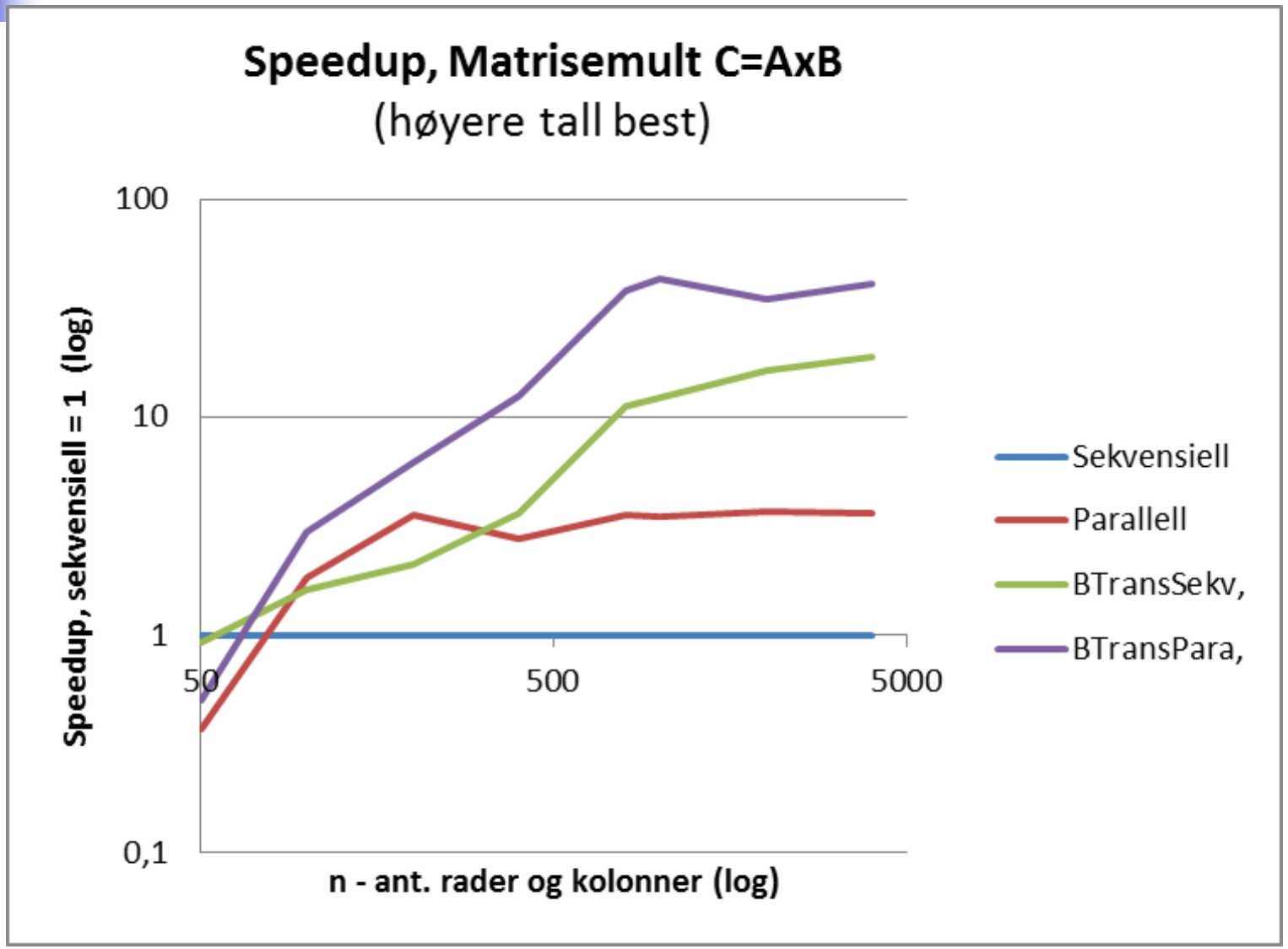
- Hukommelses-systemet i en multicore CPU ,Intel Core i5-459 3.3 GHz, – mange lag (typisk aksesstid i instruksjonssykler):

1. Registre i kjernen (1) – 8/32 registre
2. L1 cache (3-4) – 32 Kb
3. L2 cache (13) – 256 kb
4. L3 cache (32) – 8Mb
5. Hovedhukommelsen (virtuell hukommelse) (ca. 200) – 8-64 GB
6. Disken (15 000 000 roterende) = 5 ms – 1000 GB – 1-5 TB
   FlashDisk (ca 2 000 000 les, ca. 10 000 000 skriv) = ca. 1 ms

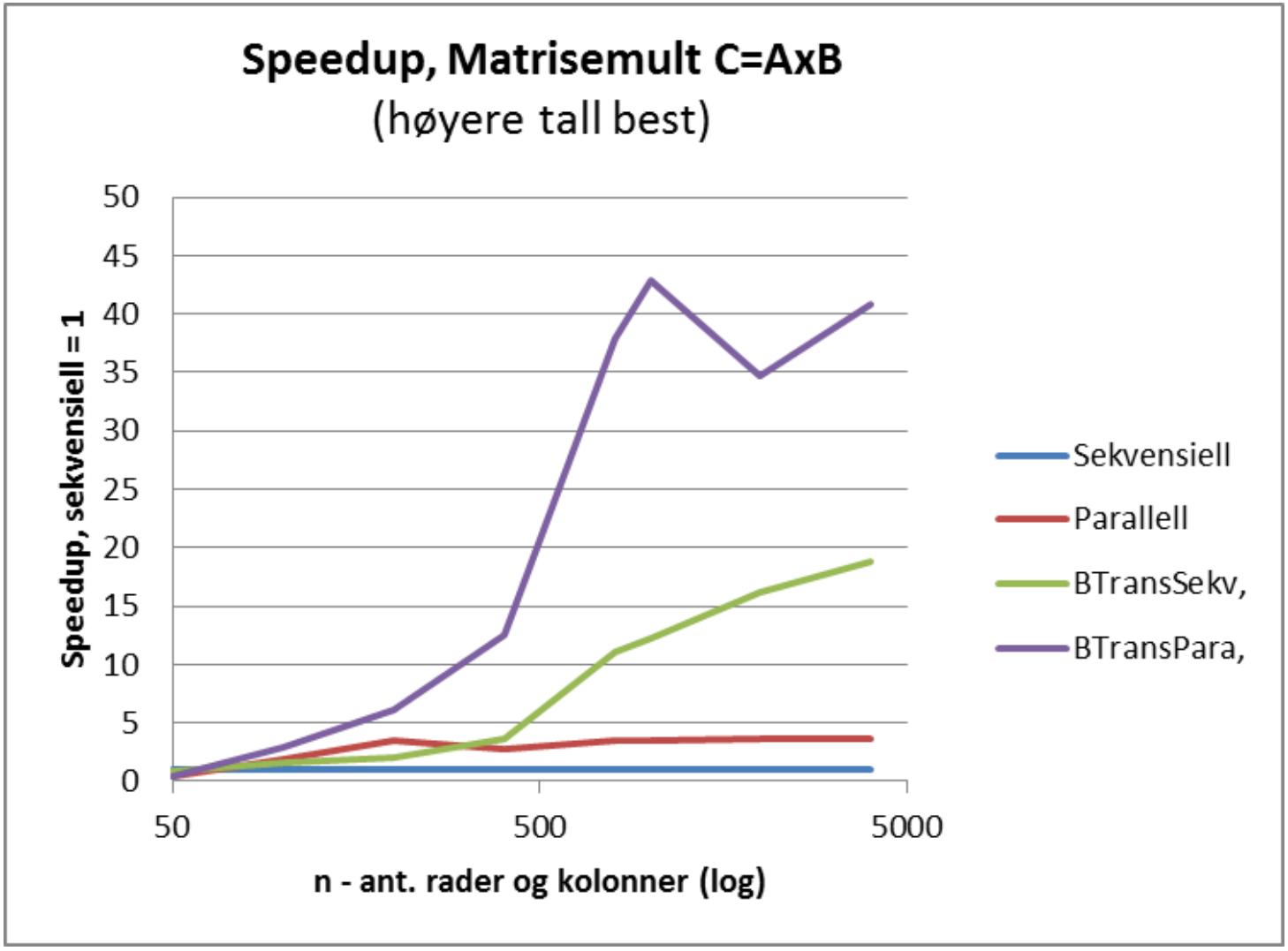
Kjøretider – i millisek. (y-aksen logaritmisk)



Kjøretidsresultater – Speedup , y-aksen logaritmisk

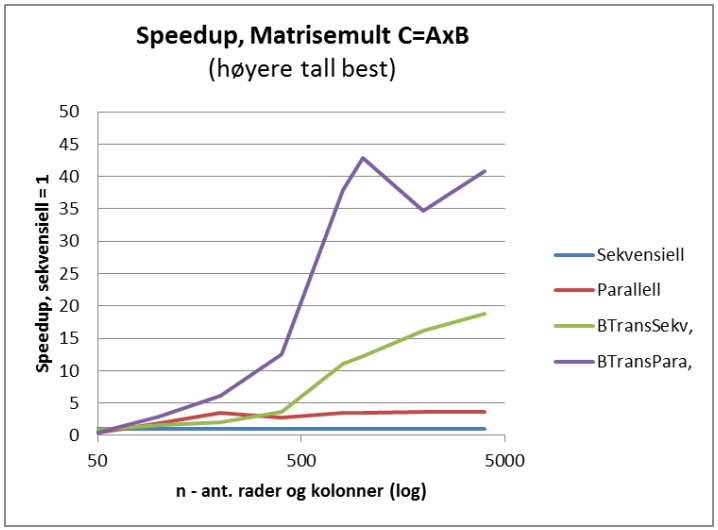


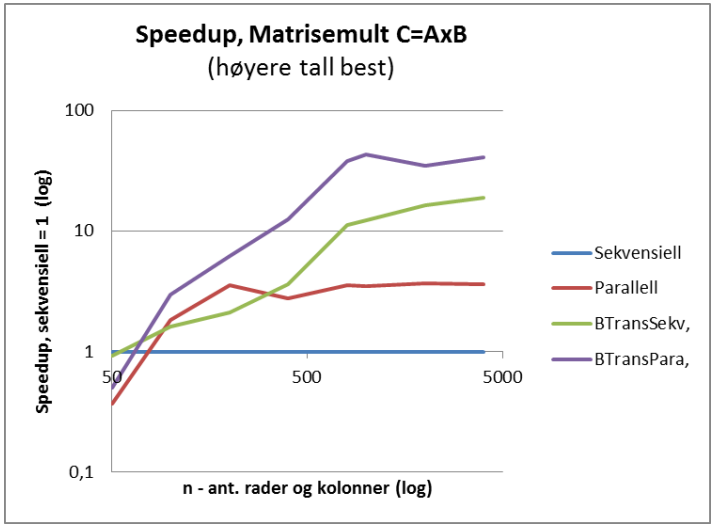
Speedup – med lineær y-akse

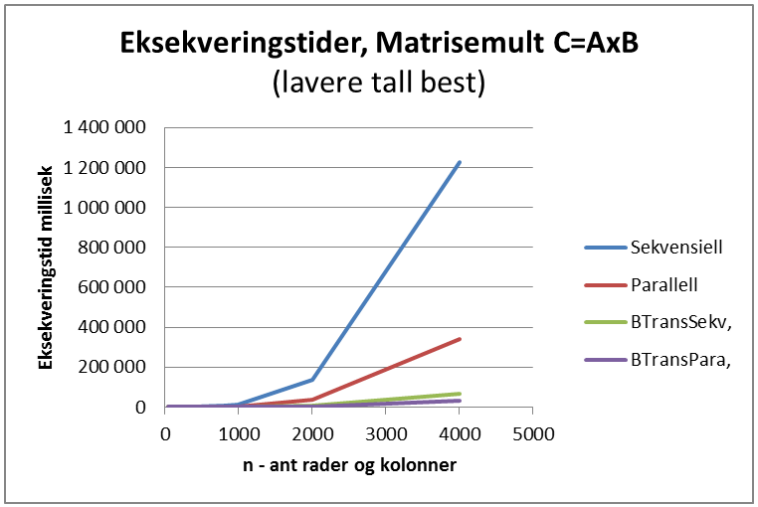


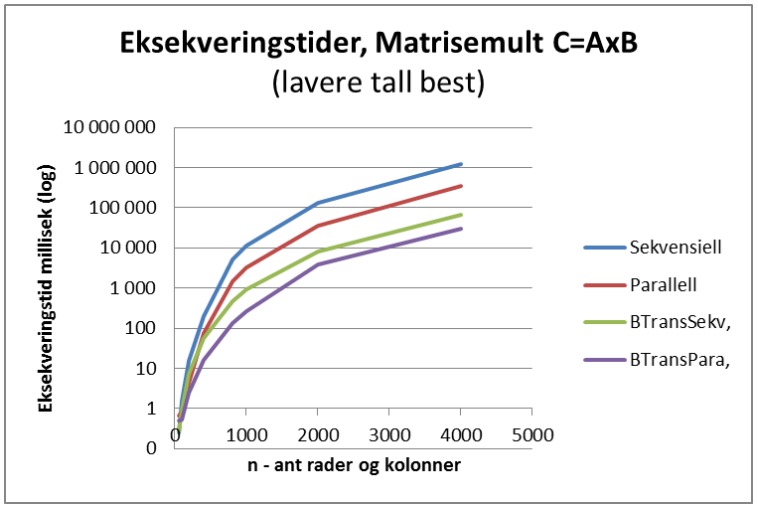
Hvordan framstille ytelse I

- Disse fire kurvene fremviser samme tall! Hvordan ?

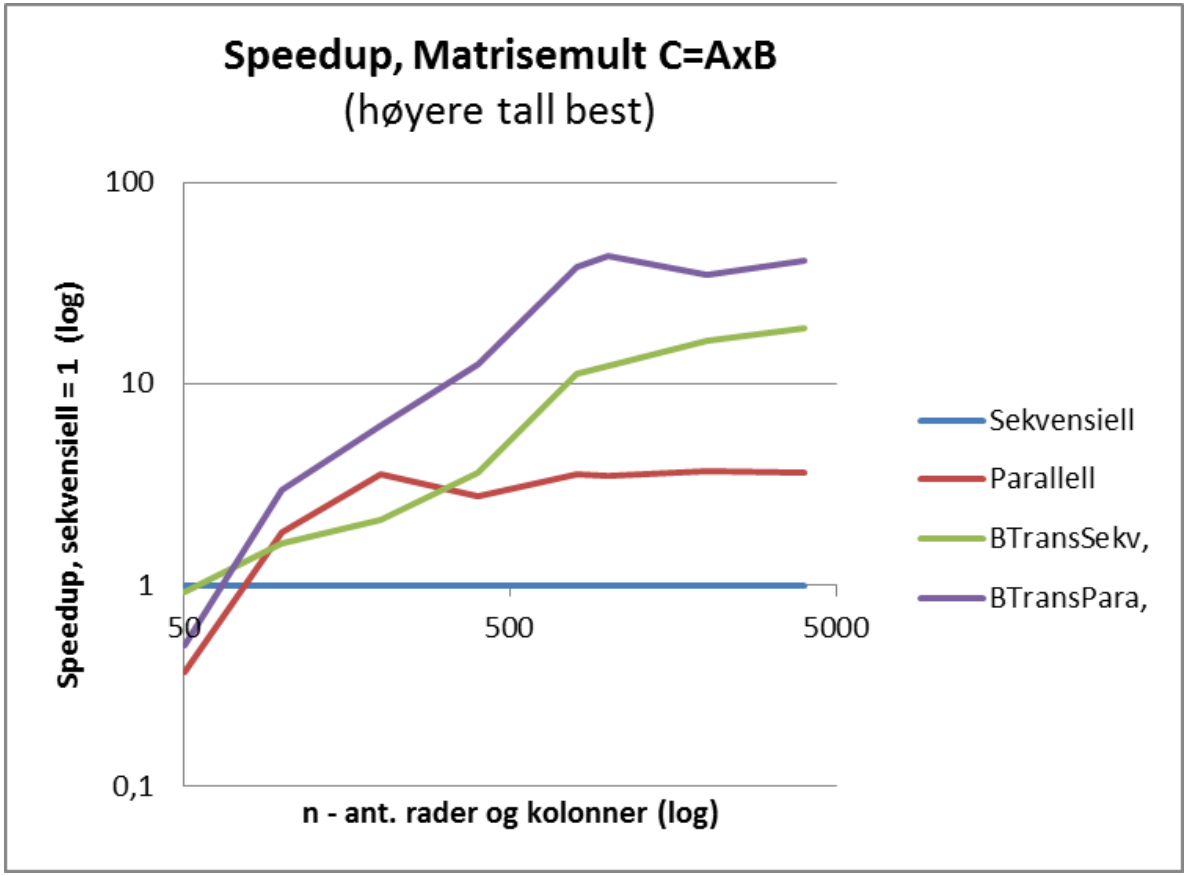






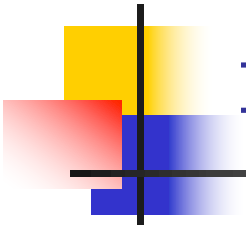


Både logaritmisk x- og y-akse



Fordel med log-akse er at den viser fram nøyaktigere små verdier, men vanskelig å lese nøyaktig mellom to merker på aksene.

III) PRAM modellen for parallelle beregninger



- PRAM (Parallel Random Access Memory) antar to ting:
  - Du har uendelig mange kjerner til beregningene
  - Enhver aksess i lageret tar like lag tid,
    - ignorerer f.eks fordelen med cache-hukommelsen
- Da blir mange algoritmer lette å beregne og programmere
- Problemet er at begge forutsetningene er helt gale.
- Det PRAM gjør er å telle antall instruksjoner utført

  Det har vi sett er helt feilaktig (Radix og Matrise-mult)
- Svært mange kurs og lærebøker er basert på PRAM

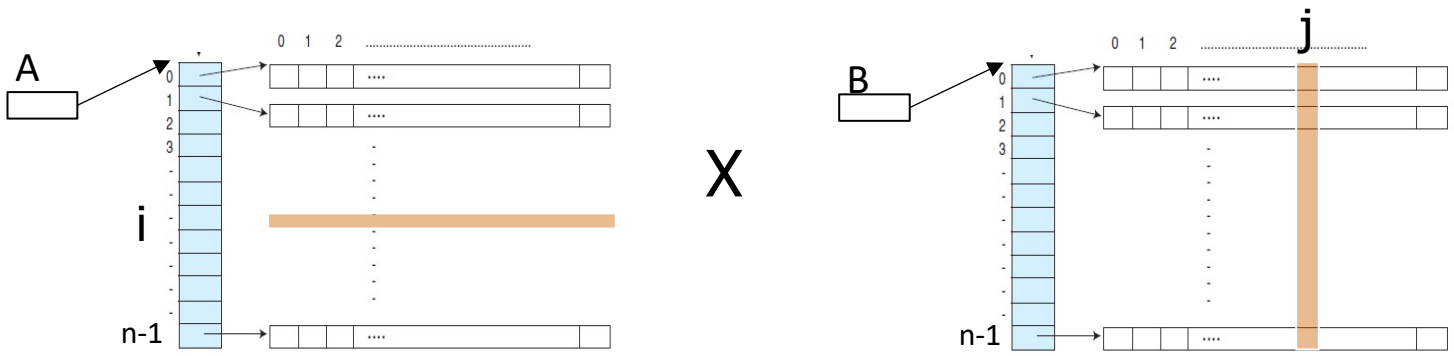
- PRAM vil si at de to sekvensielle algoritmene (med og uten transponering) gikk den utransponerte fortest!
- Dette kurset bruker **ikke** PRAM-modellen!

II) Oblig 2: matrisemultiplisering

- Matriser er todimensjonale arrayer
- Skal beregne C=AxB (A,B,C er matriser)

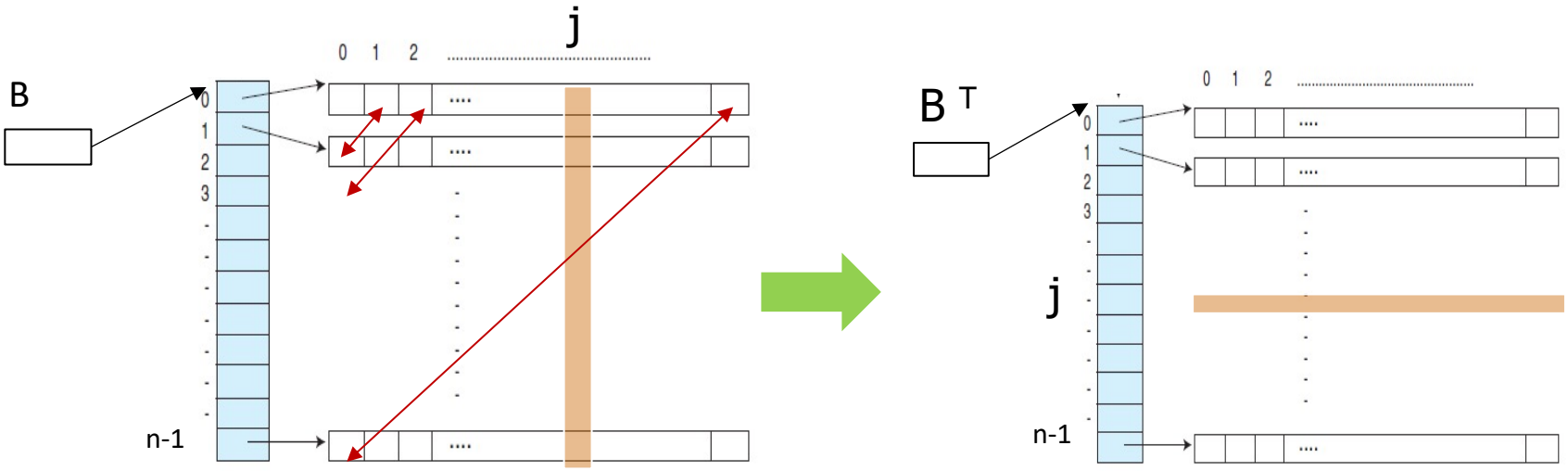
$$c[i][j] = \sum_{k=0}^{n-1} a[i][k] * b[k][j]$$

$C_{i,j} =$



Idé – transponer B (=bytte om rader og kolonner)

- Bytt om elementene i B ($B_{i,j}$ byttes om med $B_{j,i}$)
- Da blir kolonnene lagret radvis.

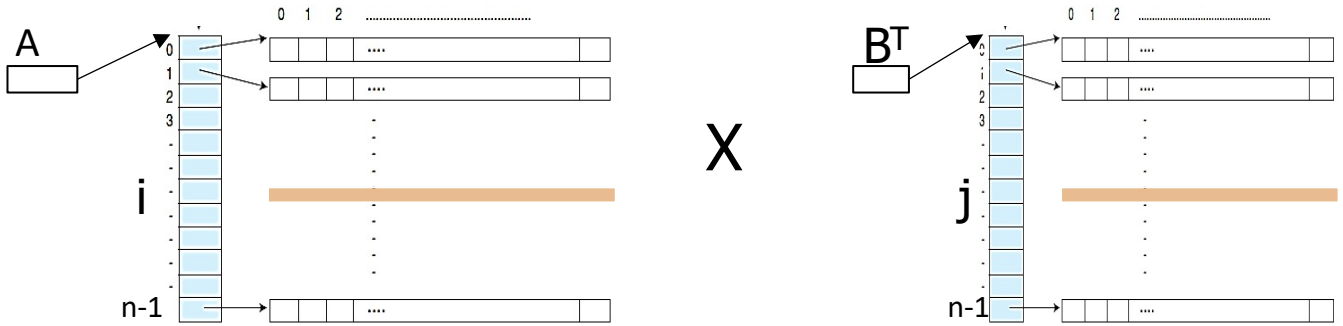


Begrunnelse: Det blir for mange cache-linjer (hver 64 byte) fra B i cachene

Vi har da at litt ny formel for C

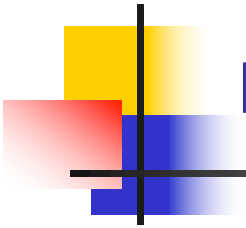
$$c[i][j] = \sum_{k=0}^{n-1} a[i][k] * b^T[j][k]$$

$C_{i,j} =$



Vi får multiplisert to rader med hverandre – går det fortere ?

Konklusjon – Matrisemultiplikasjon



- En rett frem implementasjon gikk ikke særlig fort (når n= 1000), tok det :
  - 11,24 sek med vanlig sekvensiell løsning
  - 3,24 sek. med rett fram parallellisering
  - 0,91 sek med sekvensiell + transponering av B
  - 0,26 sek med først transponering av B, så parallellisert
- Det viktigste når vi skal parallelliser er:
  - Ha den 'beste' sekvensielle algoritmen
  - Så kan du parallellisere den
- Hvis du er venner med cachen, vil parallellisering av en slik algoritme i tillegg nesten alltid lønne seg
- Dette er ett eksempel på at det å aksessere data fortløpende, ikke på tvers av data , kan gi en speedup på 10-100 ganger.
- (bare multiplikasjonen – ikke transponeringen, ble parallellisert)