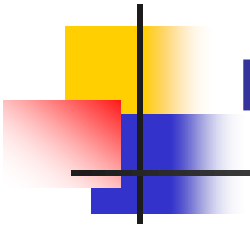


IN3030 L05v25 – Prime Numbers

Eric Jul
Programming Technology Group
Department of Informatics
University of Oslo

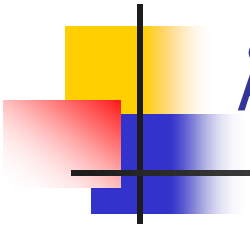


Plan for F05v25

I. Paralelisering hints
II. Amdahl's laws
III. Primtall
IV. Tidtagning
    I. JIT compilation
    II. Operativsystem?
    III. Søppel/Garbage Collection

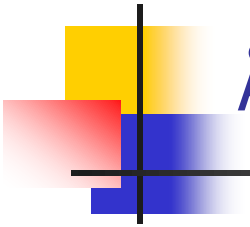
Om å parallelliser et problem

- Utgangspunkt: Vi har en sekvensiell effektiv og riktig sekvensiell algoritme som løser problemet.
- Vi kan dele opp både koden og data (hver for seg?)
- Vanligst å dele opp data
  - Som oftest deler vi opp data, og lar 'hele' koden virke på hver av disse data-delene (en del til hver tråd).
  - Eks: Matriser
    - radvis eller kolonnevis oppdeling av C til hver tråd
    - Omforme data slik at de passer bedre i cachene (transponere B)
  - Rekursiv oppdeling av data ('lett')
    - Eks: Quicksort
- Også mulig å dele opp koden:
  - Alternativ Oblig3 i INF1000: Beregning av Pi (3,1415..) med 17 000 sifre med tre ArcTan-rekker
  - Primtalls-faktorisering av store tall N for kodebrekking:
    - $N = p_1 * p_2$



Å dele opp algoritmen

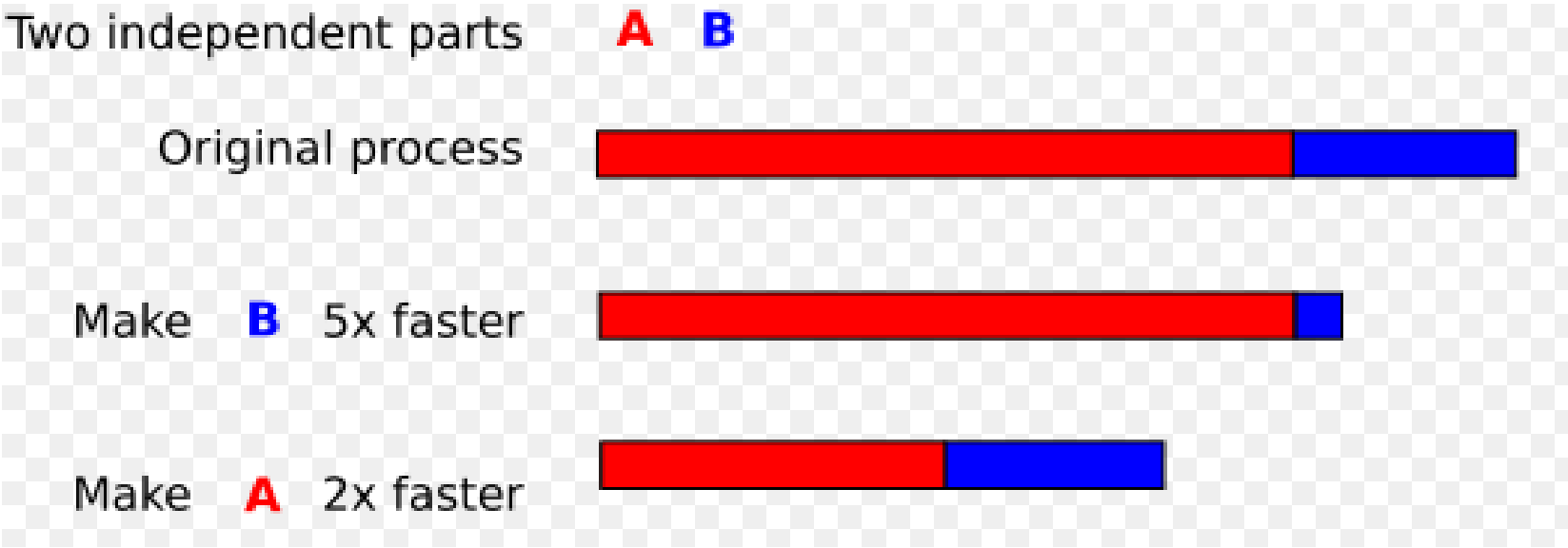
- Koden består en eller flere steg; som oftest i form av en eller flere samlinger av løkker (som er enkle, doble, triple..)
- Vi vil parallellisere med k tråder, og hver slikt steg vil få hver sin parallellisering med en CyclickBarrier-synkronisering mellom hver av disse delene + en synkronisert avslutning (join(), ..).
- Eks:
  - finnMax – hadde ett slikt steg: for (int i = 0 …n-1) -løkke
  - MatriseMult hadde ett slikt steg med trippel-løkke
  - Flere steg mulig: Eksempler senere i kurs (Radix)



Å dele opp data – del 2

- For å planlegge parallellisering av ett slikt steg må vi finne:
  - Hvilke data i problemet er lokale i hver tråd?
  - Hvilke data i problemet er felles/delt mellom trådene?
- Viktig for effektiv parallell kode.
  - Hvordan deler vi opp felles data (om mulig)
  - Kan hver tråd beregne hver sin egen, disjunkte del av data
  - Færrest mulig synkroniseringer (de tar 'mye' tid)

Amdahl – Important to parallelize big part



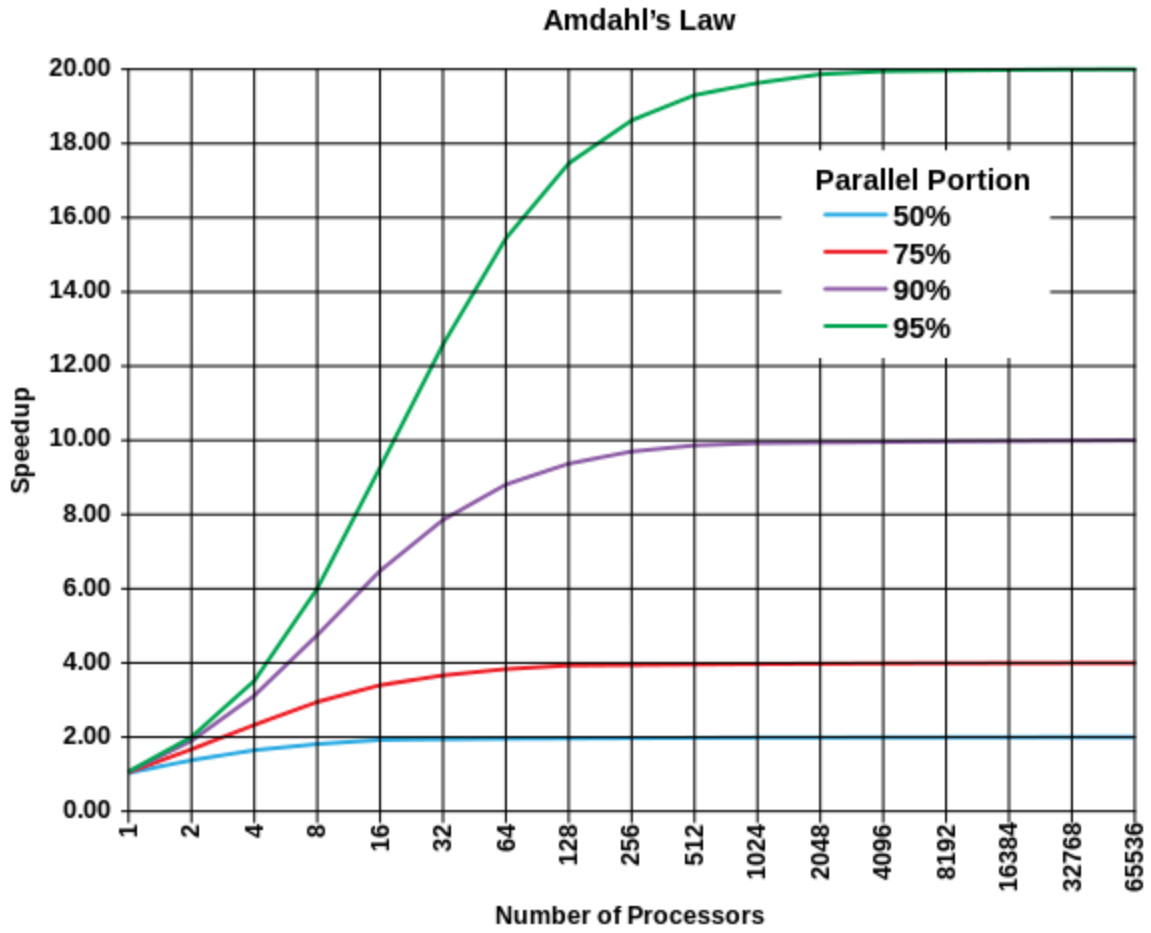
Amdahl lov for parallelle beregninger

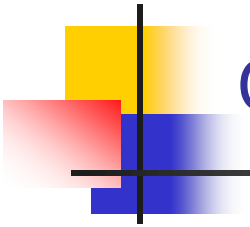
- «Hvis du først har brukt 10% av tida på en sekvensiell del, så kan resten av programmet ikke gå fortere enn 0.00 sekunder uansett hvor mange prosessorer du bruker på det. Dvs. at speedup ≤ 10»
- Amdahl lov: Har du **seq** andel sekvensiell kode og da **p** andel parallelliserbar kode i et parallelt program, **seq+p=1**, er den største speedup S du kan få med k kjerner:

$$S = \frac{tid(sekvensiell)}{tid\ (parallell)} = \frac{1}{seq+p/k} = \frac{1}{1-p+p/k}$$

- Når k $\rightarrow\ \infty$, vil S $\rightarrow \frac{1}{1-p}$.
- Er p=0.9, så er S ≤ 10 uansett hvor mange kjerner du har, og har du 'bare' 50, er S = $\frac{1}{1-0.9+0,9/50}$ = 8,5.
- Amdahls lov er pessimistisk- antar fast størrelse på problemet

Amdahl for ulike verdier av p





Om primtall

- Primtall og faktorisering av ikke-primtall.
- Et primtall er:
  Et heltall som bare lar seg dividere med 1 og seg selv.
  - 1 er ikke et primtall (det mente mange på 1700-tallet, og noen mener det fortsatt)

Om primtall og faktorisering av heltall

- Ethvert heltall N > 1 lar seg faktorisere som et produkt av primtall:
  - $N = p_1 * p_2 * p_3 * \ldots\ldots\ldots * p_k$
  - Denne faktoringen er entydig (pånær rækkefølge)
  - gjøres entydig hvis tall i faktoriseringen sorteres
  - Hvis det bare er ett tall i denne faktoriseringen, er N selv et primtall
- Eksempler:
  - 2 = 2
  - 3 = 3
  - 4 = 2*2
  - 5 = 5
  - 6 = 2 * 3 = 3 * 2
  - 7 = 7
  - 8 = 2 * 2 * 2

2 måter å lage primtall

Ønsker at finne alle primtal $p_i < N$

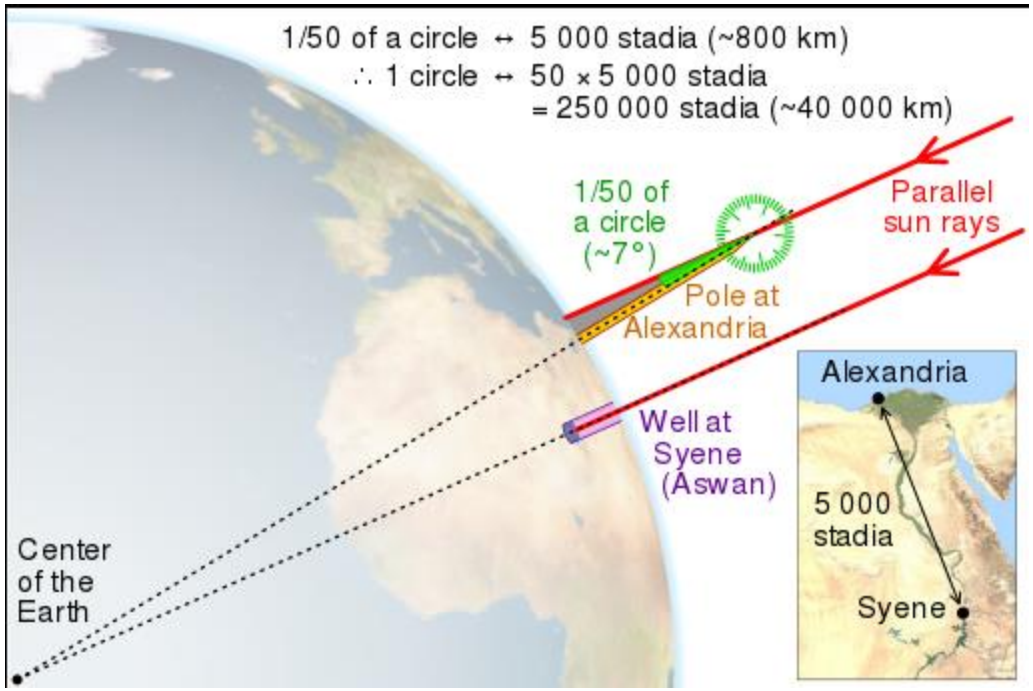
- Dividere alle tall < N med alle tall < N
  - Divisjonsmetoden
  - Bare oddetall (2 spesiel)
  - Bare opp til $\sqrt{N}$ -- hvorfor?
  - Bare primtall opp til $\sqrt{N}$ -- hvorfor?

- Lage en tabell over alle de primtallene vi trenger
  - Eratosthene sil

Litt mer om Eratosthenes

Eratosthenes, matematikker, laget også et estimat på jordas radius som var < 1,5% feil, grunnla geografi som fag, fant opp skuddårsdagen + at han var sjef for Biblioteket i Alexandria (den tids største forskningsinstitusjon).





Hvad er raskest?

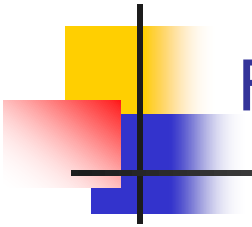
- A) Med Eratosthenes sil:

```
Z:\INF2440Para\Primtall>java PrimtallESil 2000000000
 max primtall m:2000000000
Genererte alle primtall <= 2000000000  paa     18 949 millisek
med Eratosthenes sil  og det største primtallet er:1999999973
```

- Med gjentatte divisjoner

```
Z:\INF2440Para\Primtall>java PrimtallDiv 2000000000
Genererte alle primtall <=2000000000  paa   1 577 302  millisek med
 divisjon , og det største primtallet er:1999999973
```

- Å lage primtallene p og finne dem ved divisjon (del på alle oddetall < SQRT(p), p = 3,5,7,..) er ca. 100 ganger langsommere enn Eratosthenes avkryssings-tabell (kalt Eratosthenes sil).



Finne primtall -- Eratosthenes sil

- Hvordan?
- (Blackboard)

Om primtall og faktorisering av heltall

- Ethvert heltall N > 1 lar seg faktorisere som et produkt av primtall:
  - $N = p_1 * p_2 * p_3 * \ldots\ldots\ldots * p_k$
  - Denne faktoringen er entydig (pånær rækkefølge)
  - gjøres entydig hvis tall i faktoriseringen sorteres
  - Hvis det bare er ett tall i denne faktoriseringen, er N selv et primtall
- Eksempel: faktorisering av 532
  - 532 = 2 * 266
  - 532 = 266*2 = 2 * 2 * 133
  - 532 = 266*2 = 2 * 2 * 7 * 19

Å lage og lagre primtall (Erotosthenes sil)

- Som en bit-tabell (1- betyr primtall, 0-betyr ikke-primtall)
  - Påfunnet i jernalderen av Eratosthenes (ca. 200 f.kr)
  - Man skal finne alle primtall < M
  - Man finner da de første primtallene og krysser av alle multipla av disse (N.B. dette forbedres/endres senere):
    - Eks: 3 er et primtall, da krysses 6, 9,12,15,.. Av fordi de alle er ett-eller-annet-tall (1,2,3,4,5,..) ganger 3 og følgelig selv ikke er et primtall. 6=2*3, 9 = 3*3,
      12 =2*2*3, 15 = 5*3, ..osv
    - De tallene som *ikke blir* krysset av, når vi har krysset av for alle primtallene vi har, er primtallene
- Vi finner 5 som et primtall fordi, etter at vi har krysset av for 3, finner første ikke-avkryssete tall: 5, som da er et primtall (og som vi så krysser av for, …finner så 7 osv)

Litt mer om Eratothenes sil

- Vi representerer ikke partallene på den tallinja som det krysses av på fordi vi vet at 2 er et primtall (det første) og at alle andre partall er ikke-primtall.
- Har vi funnet et nytt primtall p, for eksempel. 5, starter vi avkryssingen for dette primtallet først for tallet p*p (i eksempelet: 25), men etter det krysses det av for p*p+2p, p*p+4p,.. (i eksempelet 35,45,55,…osv.). Grunnen til at vi kan starte på p*p er at alle andre tall t < p*p slik det krysses av i for eksempel Wikipedia-artikkelen har allerede blitt krysset av andre primtall < p.
- Det betyr at for å krysse av og finne alle primtall < N , behøver vi bare å krysse av på denne måten for alle primtall p ≤ sqrt(N). Dette sparer svært mye tid.

Vise at vi trenger bare primtallene <10 for å finne alle primtall < 100, avkryssing for 3 (3*3, 9+2*3,9+4*3, ….)

1	**3**	**5**	**7**	9
11	13	15	17	19
21	23	25	27	29
31	33	35	37	39
41	43	45	47	49
51	53	55	57	59
61	63	65	67	69
71	73	75	77	79
81	83	85	87	89
91	93	95	97	99

1	**3**	**5**	**7**	**9**
11	13	**15**	17	19
21	23	25	**27**	29
31	**33**	35	37	**39**
41	43	**45**	47	49
51	53	55	**57**	59
61	**63**	65	67	**69**
71	73	**75**	77	79
81	83	85	**87**	89
91	**93**	95	97	**99**

Avkryssing for 5 (starter med 25, så 25+2*5, 25+4,5,..):

1	3	5	7	9
11	13	15	17	19
21	23	25	27	29
31	33	35	37	39
41	43	45	47	49
51	53	55	57	59
61	63	65	67	69
71	73	75	77	79
81	83	85	87	89
91	93	95	97	99

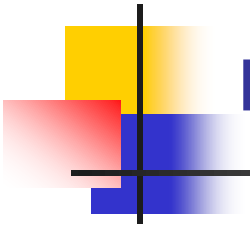
1	3	5	7	9
11	13	15	17	19
21	23	25	27	29
31	33	35	37	39
41	43	45 45	47	49
51	53	55	57	59
61	63	65	67	69
71	73	75 75	77	79
81	83	85	87	89
91	93	95	97	99

Avkryssing for 7 (starter med 49, så 49+2*7,49+4*7,.):

1	3	5	7	9
11	13	15	17	19
21	23	25	27	29
31	33	35	37	39
41	43	45 45	47	49
51	53	55	57	59
61	63	65	67	69
71	73	75 75	77	79
81	83	85	87	89
91	93	95	97	99

1	3	5	7	9
11	13	15	17	19
21	23	25	27	29
31	33	35	37	39
41	43	45 45	47	49
51	53	55	57	59
61	63 63	65	67	69
71	73	75 75	77	79
81	83	85	87	89
91	93	95	97	99

Er nå ferdig fordi neste primtall vi finner: 11, så er 11*11=121 utenfor tabellen

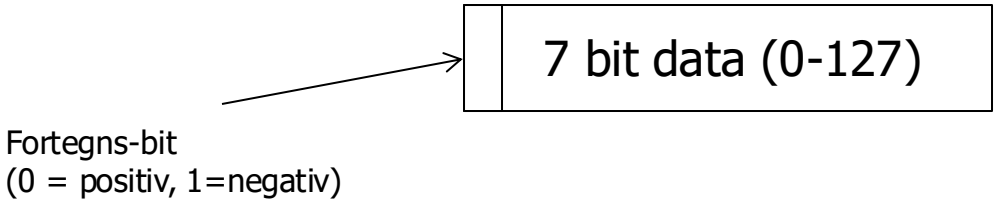


Hvordan representeres tallene?

- Kun oddetall – 2 kjenner vi!
- Array of Boolean?
  - Problem: 32 bit per primtall
- Kompakter bitarray
    - Kun 1 bit per oddetall

Hvordan bruke 8 eller 7 bit i en **byte-array** for å representere primtallene

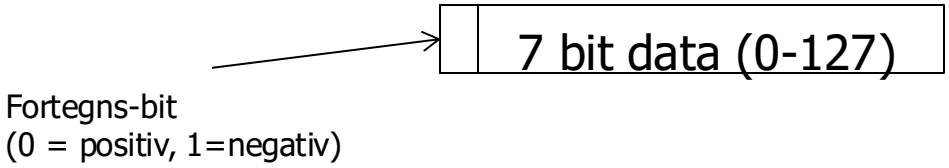
En byte = 8 bit heltall:



- Vi representer alle oddetallene (1,3,5,,,) som ett bit (0= ikke-primtall, 1 = primtall)
- Bruke alle 8 bit :
  - Fordel: mer kompakt lagring og litt raskere(?) adressering
  - Ulempe: Kan da ikke bruke verdien i byten direkte (f.eks som en indeks til en array), heller ikke +,-,* eller /-operasjonene på verdien
- Bruke 7 bit:
  - Fordel: ingen av ulempene med 8 bit
  - Ulempe: Tar litt større plass og litt langsommere(?) adressering

Hvordan representere 8 (eller 7) bit i en byte-array

byte = et 8 bit heltall



- Bruker alle 8 bitene til oddetallene:
  - Anta at vi vil sjekke om tallet ***k*** er et primtall, sjekk først om ***k*** er 2, da ja, hvis det er et partall (men ikke 2) da nei – ellers sjekk så tallets bit i byte-arrayen
    - Byte nummeret til ***k*** i arrayen er da:
      - Enten: ***k***/16, eller: ***k*** >>>4 (shift 4 høyreover uten kopi av fortegns-bitet er det samme som å dele med 16)
    - Bit-nummeret er i denne byten er da enten (***k*** %16)/2 eller (***k*** &15)>>1
  - Hvorfor dele på 16 når det er 8 bit
    - fordi vi fjernet alle partallene – egentlig 16 tall representert i første byten, for byte 0: tallene 0-15
  - Om så å finne bitverdien – se neste lysark.

Bruke 7 bit i hver byte i arrayen

- Anta at vi vil sjekke om tallet ***k*** er et primtall sjekk først om ***k*** er 2, da ja, ellers hvis det er et partall (men ikke 2) da nei – ellers:
- Sjekk da tallets bit i byte-arrayen
  - Byte nummeret til ***k*** i arrayen er da: ***k*** /14
  - Bit-nummeret er i denne byten er da: (***k*** %14)/2
- Nå har vi byte-nummeret og bit-nummeret i den byten. Vi kan da ta AND (&) med det riktige elementet i en av de to arrayene som er oppgitt i skjelett-koden og teste om svaret er 0 eller ikke.
- Hvordan sette alle 7 eller 8 bit == 1 i alle byter)
  - 7 bit: hver byte settes = 127 (men bitet for 1 settes =0)
  - 8 bit: hver byte settes = -1 (men bit for 1 settes = 0)

- Konklusjon: bruk 8 eller 7 bit i hver byte (valgfritt) i Oblig3

Seminar sessions

- The number of TA seminar sessions will be cut down from 3 to 2.
- Please fill out the Nettskjema on the course web page
- See info here: https://www.uio.no/studier/emner/matnat/ifi/IN3030/v25/beskjeder/important-poll-seminar-consolidation.html

Faktorisering av et tall M i sine primtallsfaktorer

- Vi har laget og lagret ved hjelp av Erotosthanes sil alle (unntatt 2) primtall < N i en bit-array over alle odde-tallene.
  - 1 = primtall, 0=ikke-primtall
  - Vi har krysset ut de som ikke er primtall
- Hvordan skal vi så bruke dette til å faktorisere et tall M < N*N ?
- **Svar:** Divider M med alle primtall $p_i<\sqrt{M}$ (p_i =2,3,5,..), og hver gang en slik divisjon M%p_i ==0, så er p_i en av faktorene til M. Vi forsetter så med å faktorisere ett mindre tall M'=M/p_i.
- Faktoriseringen av M = p_i *..* p_k er da produktet av alle de primtall som dividerer M uten rest.
- HUSK at en p_i kan forekommer flere ganger i svaret. eks: 20= 2*2*5, 81 = 3*3*3*3, osv
- Finner vi ingen faktorisering av M, dvs. ingen $p_i \leq \sqrt{M}$ som dividerer M med rest == 0, så er M selv et primtall.

Hvordan parallellisere faktorisering ?

1. Gjennomgås neste uke - denne uka viktig å få på plass en effektiv sekvensiell løsning med om lag disse kjøretidene for N = 2 mill:

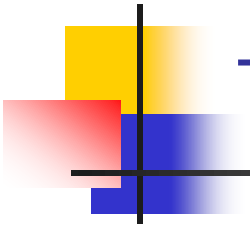
```
M:\INF2440Para\Primtall>java PrimtallESil 2000000
 max primtall m:2 000 000
Genererte primtall <= 2000000  paa        15.56 millisek
med Eratosthenes sil (  0.00004182 millisek/primtall)
……
3999998764380 = 2*2*3*5*103*647248991
3999998764381 = 37*108108074713
3999998764382 = 2*271*457*1931*8363
3999998764383 = 3*19*47*1493093977
3999998764384 = 2*2*2*2*2*7*313*1033*55229
3999998764385 = 5*13*59951*1026479
3999998764386 = 2*3*3*31*71*100964177
3999998764387 = 1163*1879*1830431
3999998764388 = 2*2*11*11*17*23*293*72139
 100 faktoriseringer beregnet paa:     422.0307ms -
 dvs:      4.2203ms. per faktorisering
```

Faktorisering av store tall med 18-19 desimale sifre

```
Uke5>java PrimtallESil 2140000000
max primtall m:2 140 000 000
bitArr.length:133 750 001
Genererte primtall <= 2 140 000 000   paa    11030.36 millisek
med Eratosthenes sil ( 0.00010530 millisec/primtall)
 antall primtall < 2 140 000 000 er: 104 748 779, dvs:  4.89% ,
 og det største primtallet er: 2 139 999 977

4 579 599 999 999 999 900 = 2*2*3*5*5*967*3673*19421*221303
4 579 599 999 999 999 901 = 457959999999999901
4 579 599 999 999 999 902 = 2*22897999999999951
4 579 599 999 999 999 903 = 3*31*13188589*3733758839
4 579 599 999 999 999 904 = 2*2*2*2*2*19*71*106087842846553
4 579 599 999 999 999 905 = 5*7*1308457142857142 83
.............
4 579 599 999 999 999 997 = 11*4163272727272727
4 579 599 999 999 999 998 = 2*121081*18911307306679
4 579 599 999 999 999 999 = 3*17*19*6625387*713333333
100 faktoriseringer beregnet paa:  333481.4427ms
 dvs:   3334.8144ms. per faktorisering

 largestLongFactorizedSafe: 4 579 599 841 640 001 173= 2139999949*2139999977
```

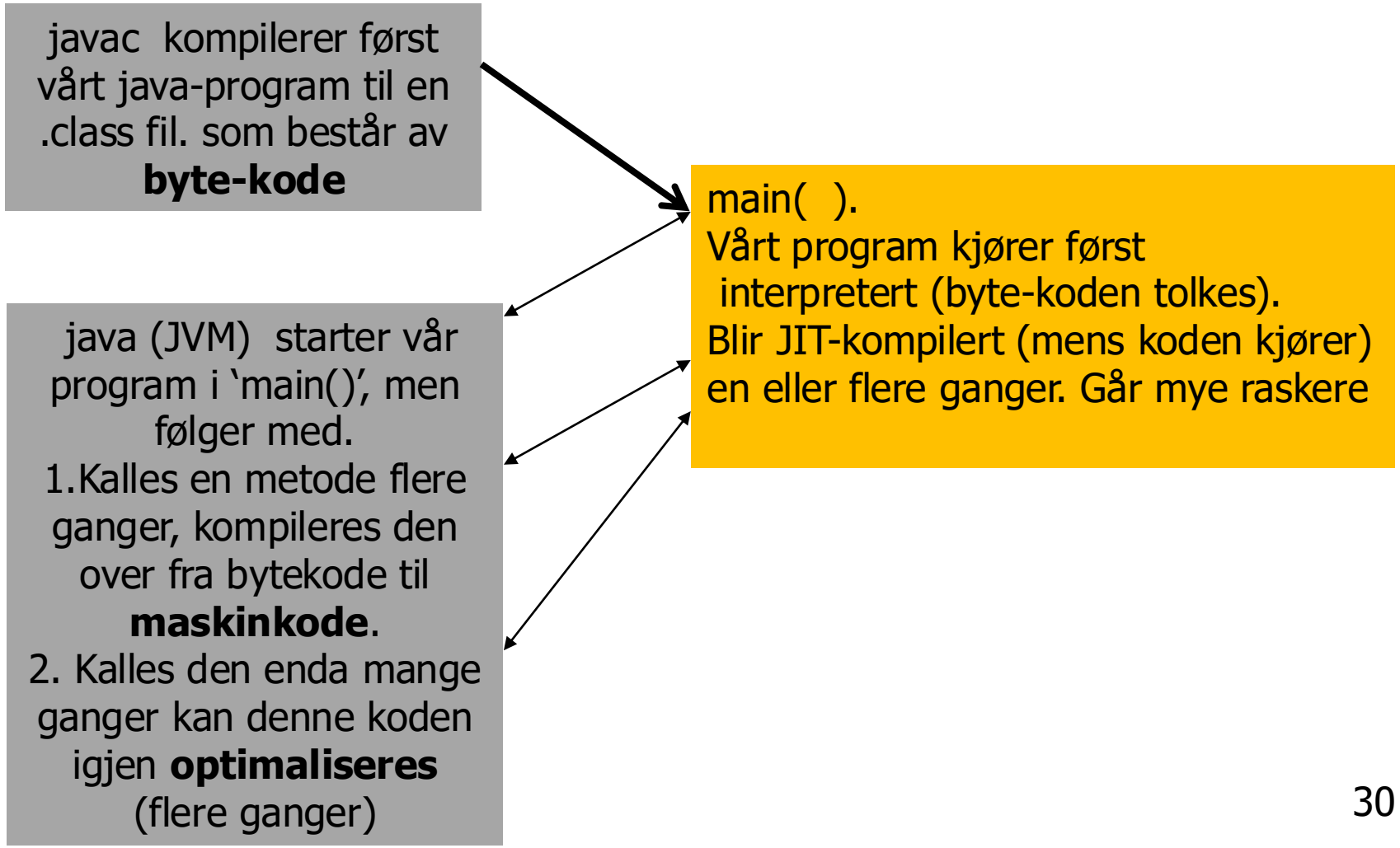


Tidtagning

- JIT –kompilering
  - Hvor mye betyr det egentlig
- Operativsystemet (Windows eller Linux)
  - Er de like raske?
- Søppeltømming i Java
  - Skjer under kjøring (med i tidene)

Tidsmålinger og JIT (Just In Time) -kompilering

- Tilbake til kompileringen av et Java-program:



Optimaliserng – ett eksempel

Original kode

```
class A {
  B b;
  public void newMethod() {
    y = b.get();
    ...do stuff...
    z = b.get();
    sum = y + z;
  }
}
class B {
   int value;
   final int get() {
      return value;
   }
}
```

1) Inline get

```
public void
newMethod() {
    y = b.value;
    ...do stuff...
    z = b.value;
    sum = y + z;
}
```

2) Fjern overflødige les

```
public  void
newMethod() {
    y = b.value;
    ...do stuff...
    z = y;
    sum = y + z;
}
```

3) Fjern overflødige variable

```
public  void
newMethod() {
    y = b.value;
    ...do stuff...
    y = y;
    sum = y + y;
}
```

4) Fjern død kode

```
public  void
newMethod() {
    y = b.value;
    ...do stuff...
    sum = y + y;
}
```

Mediantider for finnMax fra ukeoppgavene:

n= 10 000

Vi ser at kjøretidene (sekv og para) synker dramatisk fra 1.ste til neste kjøring.
Pga JIT-optimalisering

```
M:\INF2440Para\FinnMax>java FinnMaxMulti 10000 7

Kjøring:0, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     6.30 msek. , nanosek/n:  630.46
Max sekv = a:9853, paa:     0.28 msek. , nanosek/n:   28.38

Kjøring:1, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.57 msek. , nanosek/n:   56.87
Max sekv = a:9853, paa:     0.27 msek. , nanosek/n:   26.95

Kjøring:2, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.35 msek. , nanosek/n:   35.07
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    1.36

Kjøring:3, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.57 msek. , nanosek/n:   56.87
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    0.66

Kjøring:4, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.43 msek. , nanosek/n:   43.47
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    1.33

Kjøring:5, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.49 msek. , nanosek/n:   49.20
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    1.36

Kjøring:6, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.48 msek. , nanosek/n:   47.84
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    1.43

Median seq time:   0.014, median para time:   0.569,
 Speedup:    0.03, n = 10000
```

```
M:\INF2440Para\FinnMax>java FinnMaxMulti 10000000 5

Kjøring:0, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:    14.08 msek. , nanosek/n:    1.41
Max sekv = a:9999216, paa:     6.98 msek. , nanosek/n:    0.70

Kjøring:1, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:     3.17 msek. , nanosek/n:    0.32
Max sekv = a:9999216, paa:     4.75 msek. , nanosek/n:    0.47

Kjøring:2, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:     2.79 msek. , nanosek/n:    0.28
Max sekv = a:9999216, paa:     5.04 msek. , nanosek/n:    0.50

Kjøring:3, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:     2.87 msek. , nanosek/n:    0.29
Max sekv = a:9999216, paa:     5.05 msek. , nanosek/n:    0.51

Kjøring:4, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:     2.92 msek. , nanosek/n:    0.29
Max sekv = a:9999216, paa:     5.03 msek. , nanosek/n:    0.50

Median seq time:   5.052, median para time:   3.173,
 Speedup:    1.59, n = 10 000 000
```

n= 10 mill

```
M:\INF2440Para\FinnMax>java -Xint FinnMaxMulti 10000000 5

Kjøring:0, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:    67.24 msek. , nanosek/n:    6.72
Max sekv = a:9999216, paa:   179.40 msek. , nanosek/n:   17.94

Kjøring:1, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:    64.00 msek. , nanosek/n:    6.40
Max sekv = a:9999216, paa:   175.12 msek. , nanosek/n:   17.51

Kjøring:2, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:    51.42 msek. , nanosek/n:    5.14
Max sekv = a:9999216, paa:   176.23 msek. , nanosek/n:   17.62

Kjøring:3, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:    64.95 msek. , nanosek/n:    6.49
Max sekv = a:9999216, paa:   173.17 msek. , nanosek/n:   17.32

Kjøring:4, ant kjerner:8, antTråder:8
Max para = a:9999216, paa:    60.11 msek. , nanosek/n:    6.01
Max sekv = a:9999216, paa:   185.84 msek. , nanosek/n:   18.58

Median seq time: 179.403, median para time:  64.950,
 Speedup:    2.76, n = 10 000 000
```

JIT-kompilering avslått :
> java –Xint
…..
n= 10 mill

```
M:\INF2440Para\FinnMax>java FinnM 100000000 5
Kjoering:0, ant kjerner:8, antTraader:8
Max verdi parallell   i   a:99989305, paa: 41.913504 ms.
Max verdi sekvensiell i a:99989305, paa: 238.799921 ms.

Kjoering:1, ant kjerner:8, antTraader:8
Max verdi parallell   i   a:99989305, paa: 26.78024 ms.
Max verdi sekvensiell i a:99989305, paa: 235.431219 ms.

Kjoering:2, ant kjerner:8, antTraader:8
Max verdi parallell   i   a:99989305, paa: 27.791271 ms.
Max verdi sekvensiell i a:99989305, paa: 248.066478 ms.

Kjoering:3, ant kjerner:8, antTraader:8
Max verdi parallell   i   a:99989305, paa: 26.86283 ms.
Max verdi sekvensiell i a:99989305, paa: 236.013201 ms.

Kjoering:4, ant kjerner:8, antTraader:8
Max verdi parallell   i   a:99989305, paa: 27.755575 ms.
Max verdi sekvensiell i a:99989305, paa: 223.535073 ms.


Median sequential time:236.013201, median parallel time:27.755575,
 n=  100000000, Speedup:  8.59
```

n= 100 mill

JIT-kompilering +optimalisering

Søppel-tømming

Hva betyr dette for tidsmålingene

- Første gangen vi gjører er tiden vi måler en sum av:
  - Først litt interpretering av bytekode
  - Så oversetting(kompilering) av hyppig brukte metoder til maskinkode
  - kjøring av resten av programmet dels i maskinkode.
- Andre gang vi kjører, kan følgende skje:
  - JVM finner at noen av maskinkompilerte metodene våre må optimaliseres ytterligere
  - Kjøretiden synker ytterligere
- Tredje gang er som oftest optimaliseringa ferdig, men ytterligere optimalisering kan bli gjort
- Tidtakningen vår må endres !
- Vi kjører det sekvensielle og parallelle programmet f.eks 9 ganger i en løkke , noterer alle kjøretider i to arrayer som så sorteres og vi velger medianverdien = a[(a.length-1)/2]
- Du får aldri samme svaret to ganger – mye variasjon !!

FinnMax 3 ulike kjøringer (samme parametre , varierer antall tråder: 8, 16, 4)

Uke2>java FinnM 1000000 9
Kjøring:0**, ant kjerner:8, antTråder:8**
Max verdi parallell i a:999216, paa: 23.860968 ms.
Max verdi sekvensiell i a:999216, paa: 3.468803 ms.

Kjøring:1, ant kjerner:8, antTråder:8
Max verdi parallell i a:999216, paa: 0.311465 ms.
Max verdi sekvensiell i a:999216, paa: 0.549437 ms.
.....
Kjøring:8, ant kjerner:8, antTråder:8
Max verdi parallell i a:999216, paa: 0.422752 ms.
Max verdi sekvensiell i a:999216, paa: 0.532639 ms.

Median sequential time:0.52004,
median parallel time:0.429051,
Speedup: **1.26,** n = 1000000

Uke2>java FinnM 1000000 9
Kjøring:0, **ant kjerner:8, antTråder:16**
Max verdi parallell i a:999216, paa: 18.808946 ms.
Max verdi sekvensiell i a:999216, paa: 3.558043 ms.

Kjøring:1, ant kjerner:8, antTråder:16
Max verdi parallell i a:999216, paa: 1.847439 ms.
Max verdi sekvensiell i a:999216, paa: 0.453898 ms.
.....
Kjøring:8, ant kjerner:8, antTråder:16
Max verdi parallell i a:999216, paa: 0.502542 ms.
Max verdi sekvensiell i a:999216, paa: 0.471396 ms.

Median sequential time:0.509891,
median parallel time:0.646726,
Speedup: **0.90,** n = 1000000

Uke2>java FinnM 1000000 9
Kjøring:0, **ant kjerner:8, antTråder:4**
Max verdi parallell i a:999216, paa: 16.154151 ms.
Max verdi sekvensiell i a:999216, paa: 3.75507 ms.

Kjøring:1, ant kjerner:8, antTråder:4
Max verdi parallell i a:999216, paa: 1.280854 ms.
Max verdi sekvensiell i a:999216, paa: 0.520741 ms.

Kjøring:2, ant kjerner:8, antTråder:4
Max verdi parallell i a:999216, paa: 0.557136 ms.
Max verdi sekvensiell i a:999216, paa: 0.509191 ms.
.....
Kjøring:8, ant kjerner:8, antTråder:4
Max verdi parallell i a:999216, paa: 0.628527 ms.
Max verdi sekvensiell i a:999216, paa: 0.52354 ms.

Median sequential time:0.520741, median parallel time:0.628527,
Speedup: **0.88**, n = 1000000

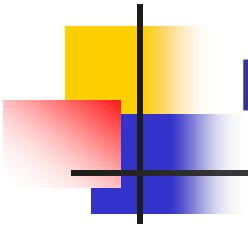
«Aldri» samme resultatet to ganger

Uke2>java FinnM 1000000 9
ant kjerner:8, antTråder:8, n = 1mill

Med antall kjøringer for median = 9
1) Speedup: **0.68**, n = 1000000
2) Speedup: 0.96, n = 1000000
3) Speedup: 0.84, n = 1000000
4) Speedup: 0.71, n = 1000000
5) Speedup: 1.06, n = 1000000
6) Speedup: 1.26**,** n = 1000000

Med antall kjøringer for median = 21
7) Speedup: 1.00, n = 1000000
8) Speedup: 0.84, n = 1000000
9) Speedup: 0.88, n = 1000000
10) Speedup: **1.75**, n = 1000000
11) Speedup: 0.87, n = 1000000
12) Speedup: 1.11, n = 1000000
13) Speedup: 1.03, n = 1000000

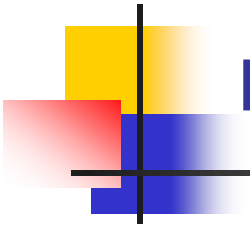
Konklusjon på JIT-kompilering



- JIT-kompilering kan skrues av med >java –Xint MittProg ..
  - Brukes bare for debugging
- JIT kompilering kan gi 10 til 30 ganger så rask eksekvering for liten n (en god del mer for stor n)
- Første, andre (og tredje) kjøring er tidsmessig sterkt misvisende
- Vi må:
  - Kjøre programmet i en løkke f.eks 9 (eller 7 eller 11) ganger
  - Legge tidene i hver sin array (sekvensielt og parallell tid)
  - Sortere arrayene
  - Ta ut medianen (element (length-1)/2), som blir vår tidsmåling

```
import java.util.concurrent.*;
import java.util.*;
class Problem2 { int [] fellesData  ; // dette er felles, delte data for alle trådene
      double [] tidene ;
      int ant, svar;
      public static void main(String [] args) {
            ( new Problem()).utfoer(args);
      }
      void utfoer (String [] args) {
              ant = new Integer(args[0]);
              fellesData = new int [ant];
              tidene = new double[9];
              for (int m = 0; m <9; m++) {
                              long tid = System.nanoTime();
                              Thread t = new Thread(new Arbeider());
                               t.start();
                               try{t.join();}catch (Exception e) {return;}
                              tidene[m] = (System.nanoTime() -tid)/1000000.0;
                              System.out.println("Tid for "+m + ", tråd:"+tidene[m]+"millisec");
              }
              Arrays.sort(tidene);
              System.out.println("Median med svar:"+svar+", for trådene:"+tidene[(tidene.length-1)/2]+" millisec");
       } // end utfoer

       class Arbeider implements Runnable {
          int i,lokalData;    // dette er lokale data for hver tråd
                  public void run() {      int sum =0;
                                           for (int i = 0; i < ant; i++) sum +=fellesData[i];
                                           svar =sum;
                  }
    } // end indre klasse Arbeider
} // end class Problem
```



Hva med operativsystemet:

- Linux og Windows har om lag like rask implementasjon av Java og trådprogrammering,
- Dag Langmyhr testet to helt like maskiner med hhv. Linux og Windows, og resultatene tidsmessig (medianer) var nesten helt like, men
  - Ulike maskiner som Ifis store servere (diamant, safir,..) har en annen Linux og en noe langsommere ytelse for korte, trådbaserte programmer.

Hva med søppeltømming – garbage collection:

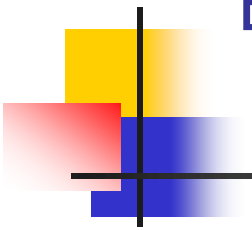
- Søppeltømming (=opprydding i lageret og fjerning av objekter vi ikke lenger kan bruke) kan slå til når som helst under kjøring:

```
Kjøring:2, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.35 msek. , nanosek/n:   35.07
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    1.36

Kjøring:3, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.57 msek. , nanosek/n:   56.87
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    0.66

Kjøring:4, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.43 msek. , nanosek/n:   43.47
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    1.33

Kjøring:5, ant kjerner:8, antTråder:8
Max para = a:9853, paa:     0.49 msek. , nanosek/n:   49.20
Max sekv = a:9853, paa:     0.01 msek. , nanosek/n:    1.36
```



End of lecture L05v25