IN3030/IN4330 Uke 7, våren 2021

Eric Jul
Programming Technology Group
Programming Section
Department of Informatics
University of Oslo
NORWAY

Hva så vi på i uke 06

1. Train Collisions in a mountain pass between Bolivia and Peru – real life synchronization problems
2. Oblig comments
3. Modellkode2-forslag for testing av parallell kode
4. Ulike løsninger på i++
5. Om primtall – Eratosthenes

Plan for uke 7

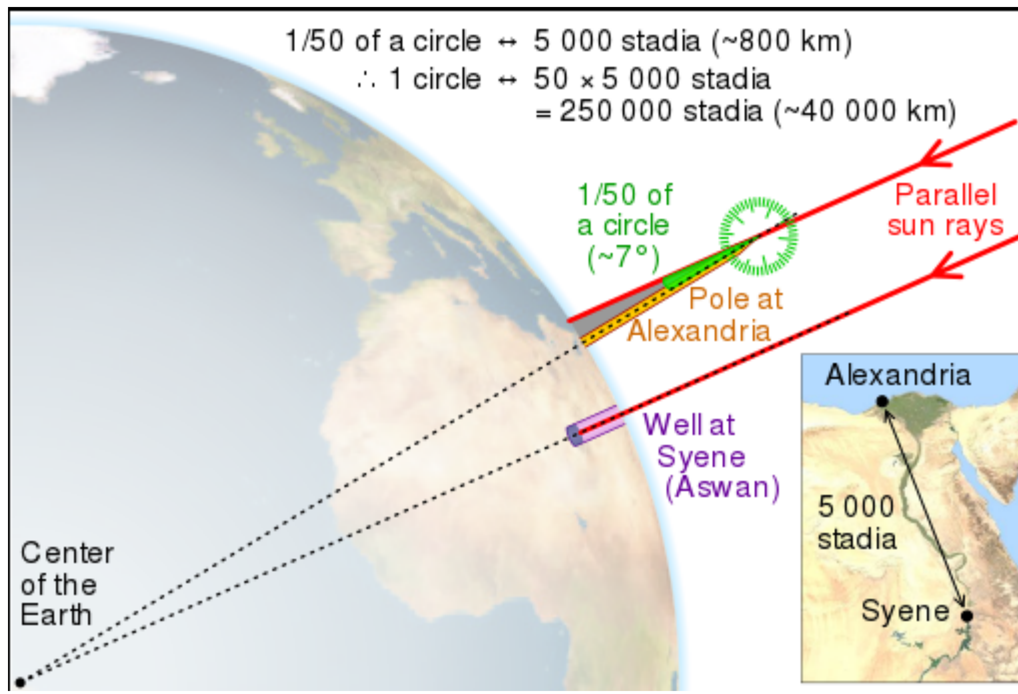
- Om primtall – Eratosthenes Sil (ES)
- Hvordan representere (ES) effektivt i maskinen
- Faktorisering av tall
- Java Measurement Harness

Om primtall – og om Eratosthenes sil (oblig 3)

- Oblig 3: 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 heltall (det mente mange på 1700-tallet, og noen mener det fortsatt)
- Ethvert tall 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); dvs. den eneste faktoriseringen av N – gjøres entydig hvis tall i faktoriseringen sorteres
  - Hvis det bare er ett tall i denne faktoriseringen, er N selv et primtall

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).



Finne primtall -- Eratosthenes sil

- Hvordan?

2 måter å lage primtall

Ønsker at finne alle primtal $p_i < N$

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

- Dividere alle tall < N med alle oddetall < $\sqrt{N}$?
  - Divisjonsmetoden
  - (Hvorfor ikke oddetallmopp til N?)

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).

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**

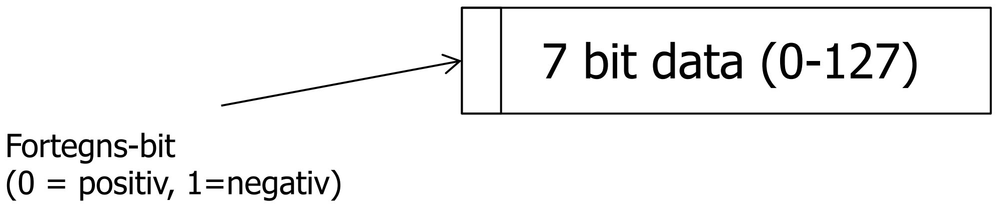
Er nå ferdig fordi neste primtall vi finner: 11, så er 11*11=121 utenfor tabellen

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

7 bit data (0-127)

Fortegns-bit
(0 = positiv, 1=negativ)

- 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 arryene 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

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 = 4579599999999999901
4 579 599 999 999 999 902 = 2*2289799999999999951
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*130845714285714283
.............
4 579 599 999 999 999 997 = 11*416327272727272727
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
```

Oblig 3: Primtall

End of lecture uke07 2021