INF2440 Uke 11, v2014 om parallell debugging og Goldbachs problem, om Oblig 3

Arne Maus
OMS,
Inst. for informatikk

Fra hjemmesida til INF2440:

Plan for resten av semesteret:

Forelesninger: 28.mars (i dag), 4.april, 25.april og 2.mai.
(IKKE 18. april som er Langfredag).

Oblig 3: Fullføring av parallell Radix-sortering (2 siffer) -
frist 25. april. Tekst kommer snart.

Prøveeksamen: 26. mai. (mer info senere)

Eksamen: 2. juni kl. 14.30

Hva så vi på i Uke10

- En kommentar om Oblig2
- Automatisk parallellisering av rekursjon
- PRP- Parallel Recursive Procedures
  - Nåværende løsning (Java, multicore CPU, felles hukommelse) – implementasjon: Peter L. Eidsvik
- Demo av to eksempler med kjøring
- Hvordan ikke gå i fella: Et Rekursivt kall = En Tråd
  - Halvautomatisk oversettelse (hint/kommandoer fra kommentarer)
- Hvordan virker en kompilator (preprosessor) for automatisk parallellisering
  - Prinsipper (bredde-først og dybde-først traversering av r-treet)
  - Datastruktur
  - Eksekvering
- Krav til et program som skal bruke PRP
- Slik bruker du PRP – Ukeoppgave neste uke.

Hva skal vi se på i Uke 11

I) En del sluttkommentarer og optimalisering av Oblig2

II) Debugging av parallelle programmer

III) Et større problem – uløst problem siden 1742

- Vil du tjene $1 mill. ?
- Løs Goldbach's påstand (alternativt: motbevis den)
- Formulering og skisse av løsning av tre av Goldbachs problemer
- Parallellisering av disse (ukeoppgave neste uke)

IV) Om Oblig 3

I) Om speedup av Oblig2; 'optimal' sekvensiell og parallell kode:

	n	sekv(ms)	para(ms)	Speedup	PerElemSek	PerElemPar
EratosthesSil	2000000000	14131.98	7060.41	2.00		
Faktorisering	2000000000	24586.18	15845.43	1.55	245.8618	158.4543
Total tid	2000000000	38613.92	22938.85	1.68		
EratosthesSil	200000000	1078.20	268.08	4.02		
Faktorisering	200000000	5970.54	2817.03	2.12	59.7054	28.1703
Total tid	200000000	7050.84	3085.10	2.29		
EratosthesSil	20000000	68.80	17.71	3.88		
Faktorisering	20000000	683.02	407.84	1.67	6.8302	4.0784
Total tid	20000000	747.09	425.56	1.76		
EratosthesSil	2000000	6.02	1.92	3.14		
Faktorisering	2000000	146.56	214.39	0.68	1.4656	2.1439
Total tid	2000000	152.12	216.29	0.70		
EratosthesSil	200000	1.02	0.61	1.65		
Faktorisering	200000	67.47	163.95	0.41	0.6747	1.6395
Total tid	200000	68.53	164.62	0.42		
EratosthesSil	20000	0.06	0.10	0.55		
Faktorisering	20000	14.06	23.98	0.59	0.1406	0.2398
Total tid	20000	14.12	24.20	0.58		
EratosthesSil	2000	0.01	0.27	0.05		
Faktorisering	2000	23.20	46.89	0.49	0.2320	0.4689
Total tid	2000	23.24	47.25	0.49		

Ikke-optimal, men riktig kode

```
// Sekvensiell faktorisering av primtall
ArrayList<Long> factorize (long num) {
        ArrayList <Long> fakt = new ArrayList <Long>();
        int pCand =2;
        long pCand2=4L, rest= num;

        while (rest > 1 &&  pCand2 <= num) {
                while ( num % pCand == 0){
                        fakt.add((long) pCand);
                        rest /= pCand;
                }
                pCand = nextPrime(pCand);
                pCand2 =(long) pCand*pCand;
        }
        if (rest >1) fakt.add(rest);
        return fakt;
} // end factorize
```

Tips: Hva er grensene når vi har funnet første faktor

Manglende optimalisering av sekvensiell faktorisering

	n	sekv(ms)	para(ms)	Speedup
EratosthesSil	2000000000	14209.31	7205.05	1.97
Faktorisering	2000000000	441606.92	16189.86	27.28
Total tid	2000000000	455816.24	23394.92	19.48
EratosthesSil	200000000	1107.95	245.96	4.50
Faktorisering	200000000	45323.05	2864.98	15.82
Total tid	200000000	46431.01	3110.93	14.93
EratosthesSil	20000000	66.82	19.10	3.50
Faktorisering	20000000	5745.96	365.60	15.72
Total tid	20000000	5812.79	384.70	15.11
EratosthesSil	2000000	6.13	2.13	2.87
Faktorisering	2000000	546.41	228.45	2.39
Total tid	2000000	552.55	230.58	2.40

Her har vi ikke-optimal sekvensiell kode, **men** 'optimal' parallell kode

3999999999999999899 = 107*37383177570093457
3999999999999999900 = 2*2*3*5*5*89*1447*1553*66666667
3999999999999999901 = 19*2897*72670457642207
3999999999999999902 = 2*49965473*40027640687
3999999999999999903 = 3*101*241*54777261958561
3999999999999999904 = 2*2*2*2*2*1061*117813383600377
3999999999999999905 = 5*7*13*8791208791208791
3999999999999999906 = 2*3*3*17*31*421674045962471
3999999999999999907 = 14521697*275449900931
3999999999999999908 = 2*2*11*168409*539811357523
3999999999999999909 = 3*2713*491460867428431
3999999999999999910 = 2*5*399999999999999991
3999999999999999911 = 167*659*3581*86629*117163
3999999999999999912 = 2*2*2*3*7*7*4127*669869*1230349
3999999999999999913 = 239*16736401673640167
39999999999999899 = 39999999999999899
39999999999999900 = 2*2*3*5*5*7*952381*19999999
39999999999999901 = 181*229*229*47237*89213
39999999999999902 = 2*16103*1242004595417
39999999999999903 = 3*191*69808027923211
39999999999999904 = 2*2*2*2*2*47*7867*19843*170371
39999999999999905 = 5*151*230341*230007391
39999999999999906 = 2*3*3*3*3*3*3*2089*13133002513
39999999999999907 = 7*7*7*29*4021312958681
39999999999999908 = 2*2*11*19*7621*6278295493

Noen faktoriseringer

Observasjoner :

- Hvor mange faktorer er det i de fleste?
- Hvor store er tallene?
- Deler vi talllinja av primtall (2 milliarder) på 8 kjerner får vi 250 mill. til hver tråd
- Hvilken tråd finner de fleste faktorene?

Optimalisering av parallell faktorisering:

- Ved parallellisering av faktorisering, får hver kjerne (8) sin del av primtallene (i eksempelet primtallene på 25 bit i bit-tabellen hver)
- Hvor stor sannsynlighet er det for at et vilkårlig tall k er delbart på 3 - uten rest?
- Hvor stor sannsynlighet er det for at et vilkårlig tall k er delbart på 4093 - uten rest?
- Mens vi i det sekvensielle tilfellet hadde kontroll over 'resten' av tallet har vi det **ikke** på samme måte parallell faktorisering.
- Vi må først finne det gode optimaliseringa sekvensielt, og gjøre noe av det samme med en felles variabel (AtomicLong) mellom trådene.

Hvorfor får vi ulik speedup, og bare for store verdier av n ?

	n	sekv(ms)	para(ms)	Speedup
EratosthesSil	20000000	68.80	17.71	3.88
Faktorisering	20000000	683.02	407.84	1.67
Total tid	20000000	747.09	425.56	1.76
EratosthesSil	2000000	6.02	1.92	3.14
Faktorisering	2000000	146.56	214.39	0.68
Total tid	2000000	152.12	216.29	0.70
EratosthesSil	200000	1.02	0.61	1.65
Faktorisering	200000	67.47	163.95	0.41
Total tid	200000	68.53	164.62	0.42
EratosthesSil	20000	0.06	0.10	0.55
Faktorisering	20000	14.06	23.98	0.59
Total tid	20000	14.12	24.20	0.58

- Ulik speedup på de to algoritmene:
  - Overhead:
    - startup
    - synkronisering
  - Lastballansering mellom trådene ?
  - Effekter av cachen (se begge algoritmer)?

Tips til utskrift mm

- Saml inn tidene først i en flere-dimensjonal double[][][] array, og skriv det så ut oversiktelig (funksjon, n, iterasjon)
- Faktoriseringa kan skrives ut mens du finner den.
  (kan ikke lage array av ArrayList-er.)
- Lag utskriftsmetoder som skriver ut både på fil og skjerm

```
/** for også utskrift på fil */
 synchronized void println(String s) {
          ut.outln(s);
          System.out.println(s);
 }

 /** for også utskrift på fil */
 synchronized void print(String s) {
          ut.out(s);
          System.out.print(s);
 }
```

II) Debugging – feilfjerning parallelt

- Antar at vi har et program som lar seg starte opp
- Felles problem i sekvensiell og parallell kode:
- Er det en feil her?
  - Terminerer programmet
  - Gale resultater
  - Samme feil hver gang ?
    - Et sekvensielt program kan gjøres deterministisk
      - (eks. Random r = new Random(123) vil produsere samme tall-rekke ved neste kjøring)
    - Parallelle programmer kan ikke gjøres deterministiske
      - 'Never same result twice'
  - Er feil avhengig av størrelsen på problemet som løses av programmet , av n ?

Råd1: Finn først det 'minste' eksempelet som feiler

Du har en feil som kan reproduseres på et relativt lite eksempel

- EKS fra debugging av min parallelle Oblig2-løsning (parallell ErSil + parallell faktorisering.):
  - **para: 39910 = 65*2*307**
    - NEI!! (65 = 13*5)
- Symptomer:
  - Kom ikke hver gang (bare ca. hver 4. gang))
  - Kom ikke i den sekvensielle faktoriseringa.
- Spørsmål::
  - Er det en tidsavhengig feil ?
    - Svar1 ? Sannsynligvis det
  - Eller det ‘bare’ en sekvensiell feil i den parallelle koden?
    - Svar2: Mindre sannsynlig. Nesten alle parallelle faktoriseringer er riktige (og ingen andre brukte 65 som primtall)

para: 39910 = 65*2*307 - ikke

- Er det feil i Eratosthenes bit-tabell eller i selve faktoriseringa
  - Jeg prøvde ut flere optimaliseringer.
- Sjekket først synkronisering
  - Viktig: At bare en tråd (tråd-0) initierer konstanter o.l., og genererer alle primtall < $\sqrt{200}$ =14,..., og at alle andre tråder venter på at det er ferdig.
- **Råd2: Ikke** debug **ved å gå linje for linje gjennom koden** !
- **Råd3**: Bruk binær feilsøking:
  - Plasser en System.out.println(..) '**midt i koden**' og sjekk de data som der skrives ut – virker de riktige?
    - Hvis ja: feilen nedenfor, nei: feilen er ovenfor
  - Her: Siden problemet feilet med n=200, så:
    - Skrev ut alle primtallene < n (200)
  - Vi hadde to algoritmer (Eratosthenes eller faktoriseringen) – var det den første eller den andre ?

Debug:2
Debug:3
Debug:5
Debug:7
Debug:11
Debug:13
Debug:17
Debug:23
Debug:29
Debug:33
Debug:39
Debug:41
Debug:47
Debug:51
Debug:53
Debug:57
Debug:59
Debug:69

Noen av Primtallene var altså gale (for mange)

- Første feil : 33 !
- Feilen var altså i Erotasthenes Sil i parallell:
- Algoritmen, skisse (for n=200, $\sqrt{n}$ =14):
  - Finn først alle primtall < $\sqrt{n}$ sekvensielt
    - Dette er de vi trenger for å lage alle under 200
  - Del tallinja 1..200 i antallKjerner (8) biter, jeg fikk:

```
left:1, right:27
left:1, right:13
left:71, right:83
left:99, right:200
left:85, right:97
left:57, right:69
left:43, right:55
left:15, right:27
left:29, right:41
```

Ikke opplagt riktig

```
Debug:2
Debug:3
Debug:5
Debug:7
Debug:11
Debug:13
Debug:17
Debug:23
Debug:29
Debug:33
Debug:39
Debug:41
Debug:47
Debug:51
Debug:53
Debug:57
Debug:59
Debug:69
```

- **Råd4:** Litt analyse (papir og blyant):
  - Denne metoden ble brukt av både sekvensiell innledende fase av parallell alle <$\sqrt{n}$ (rød) og parallell (sort):
  - Den er også tilpasset grensene for hva innholdet av en byte er:
    - byte[0] 1-13, byte[1] 15-27,…
  - Siden $\sqrt{200}$ =14 > 13 , så OK ?
  - De andre grensene så ut til å være veldig skjevt fordelt ?
  - Men siden da jeg regnet ut antall primtall per kjerne:

    (200 /8) /14)*14 = (25/14)*14 = 14

    og siste tråd tar resten, er dette OK.
  - Konklusjon: Tråd-2 (29-41) feiler med 3-er avkryssingene

left:1, right:27
left:1, right:13
left:71, right:83
left:99, right:200
left:85, right:97
left:57, right:69
left:43, right:55
left:15, right:27
left:29, right:41

Hvis den alltid tror at 65 er et primtall, hvorfor går dette ofte OK ?

Råd5: Forstå feilen

- Som oftest får vi følgende svar : 39910 = 2*5*13*307 som er helt riktig!
- Og hvorfor vil den sekvensielt alltid ha det riktig selv om den tror at 65 er et primtall?
- Se på koden:

Hvis 65 = 5*13 er to faktorer i et tall num

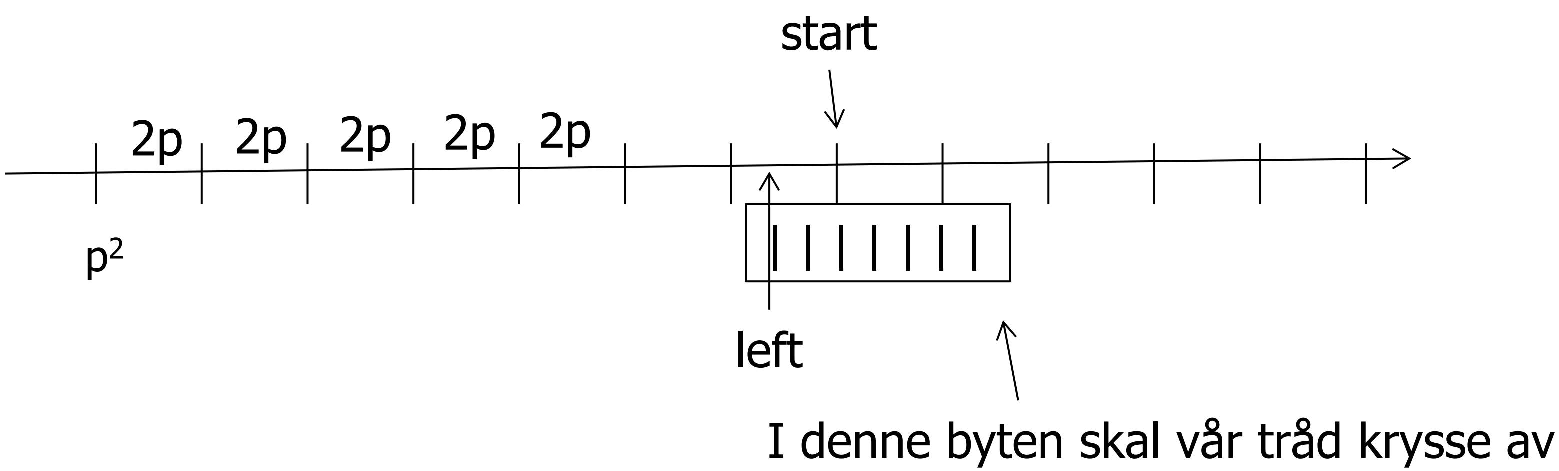
- og vi allerede har delt ned num med de mindre primtallene 5 og 13, så finner vi ikke 65 en faktor (den er dividert bort)
- Dividerer vi *først* ned med 5 og 13 (alltid ved sekvensiell), så vil vi aldri oppdage 65.
- Hvis tråd2 deler num ned før tråd0, vil vi oppdage 65 – ellers ikke.

```
// Sekvensiell faktorisering av primtall
ArrayList<Long> factorize (long num) {
        ArrayList <Long> fakt = new ArrayList <Long>();
        int pCand =2;
        long pCand2=4L, rest= num;

        while (rest > 1 &&  pCand2 <= num) {
                while ( num % pCand == 0){
                        fakt.add((long) pCand);
                        rest /= pCand;
                }
                pCand = nextPrime(pCand);
                pCand2 =(long) pCand*pCand;
        }
        if (rest >1) fakt.add(rest);
        return fakt;
} // end factorize
```

Feilen - konklusjon:

- Diagnose – enten ble 3-ere , 5ere…..avkrysset galt i byte[1], byte[2],.., eller så har vi et timing-problem
- FASIT:
  - Hvor starter vi neste kryss fra primtall p i en byte (bit-nummeret) som har sitt venstre, minste bit i left.
  - p^2 ikke er i inneværende byte. Hva er verdien til start



start = ((left - $p * p$)/($p + p$)) $*$ ($p + p$) + $p * p$;
while(start < left) start += $p + p$;

Avsluttende bemerkninger om debuging av parallelle programmer:

- **Råd6:** Legg System.out.println(s) innn i en:
  synchronized void println(String s) i den ytre klassen
  - Da vil utskrifter ikke blandes.
- **Råd7**: Hvis ingen ting skjer når vi starter programmet:
  - Lag en 5-6 setninger av typen : print(«A»); print(«B»);..og plasser de jevnt over rundt i programmet (i toppen av løkker)
  - Da ser vi hvor langt programmet kom før det hang seg.
- Start å lete etter siste bokstav i koden som kom ut.

III) Christian Goldbachs påstand i 1742:

- Alle partall $m = n+n > 4$ kan skrives som en sum av to primtall som er oddetall.
  - $m = p1+p2 \quad (p1 \leq p2)$
  - Eks: 6 = 3+3, 14 = 7+7 og 14 = 11+3,....
- Antall slike ulike summer av to primtall for en gitt m kaller vi G(m).
- Bevis at $G(m) > 0 \ \forall \ m$.

vel egentlig..:

- Goldbach foreslo at alle tall kan skrives som sum av tre primtall. Den store matematiker Euler ryddet opp i det og utledet lett:
  - Alle oddetall kan skrives som sum av tre primtall
  - Alle partall kan skrives som summen av to primtall
- Goldbach mente selv at 1 var et primtall (naturlig nok)
- Goldbachs påstand er ikke bevist matematisk for partall.

Praktisk på datamaskiner (Wikipedia):

- T. Oliveira e Silva is running a distributed computer search that has verified the conjecture for $n \leq 4 \times 10^{18}$ (and double-checked up to 4×10^{17}). One record from this search: 3325581707333960528 needs minimal prime 9781 (=p1 : n =p1+p2))
- Dvs for de fleste tall som kan representeres i en long

Goldbachs brev til Euler hvor han foreslår at alle tall kan skrives som sum av tre primtall

haben, nicht bestehen, es wäre aber schon was sonderliches, wann dieselbe series lauter numeros unico modo in duo quadrata divisibiles gäbe. Auf solche Weise will ich auch eine conjecture hazardiren: dass jede Zahl welche aus zweyen numeris primis zusammengesetzt ist ein aggregatum so vieler numerorum primorum sey, als man will (die unitatem mit dazu gerechnet) bis auf die congeriem omnium unitatum*. zum Exempel

$$4 = \begin{cases} 1+1+1+1 \\ 1+1+2 \\ 1+3 \end{cases} \qquad 5 = \begin{cases} 2+3 \\ 1+1+3 \\ 1+1+1+2 \\ 1+1+1+1+1 \end{cases} \qquad 6 = \begin{cases} 1+5 \\ 1+2+3 \\ 1+1+1+3 \\ 1+1+1+1+2 \\ 1+1+1+1+1+1 \end{cases} \quad \&c$$

Hierauf folgen ein paar observationes so demonstriret werden können:

Si v. sit functio ipsius x. eiusmodi ut facta $v = c$. numero cuicunque, determinari possit x per c. et reliquas constantes in functione expressas, poterit etiam determinari valor ipsius x. in aequatione $v^{2x+1} = (2v+1)(v+1)^{x-1}$

Si concipiatur curva cuius abscissa sit x. applicata vero sit summa seriei $\frac{x^n}{n.2^{2n}}$ posita n. pro exponente terminorum, hoc est, applicata $= \frac{x}{1.2^2} + \frac{x^2}{2.2^4} + \frac{x^3}{3.2^6} + \frac{x^4}{4.2^8} + \&c$. dico, si fuerit abscissa $= 1$. applicatam fore $= l\frac{4}{3}$ · [et hac applicata $= y$ erit $y = l\frac{a}{a-x}$]

2 . . . . $l2$.

3 . . . . $2l2$.

4 vel major . . . . infinitam.

Ich verharre mit aller ersinnlichen Hochachtung

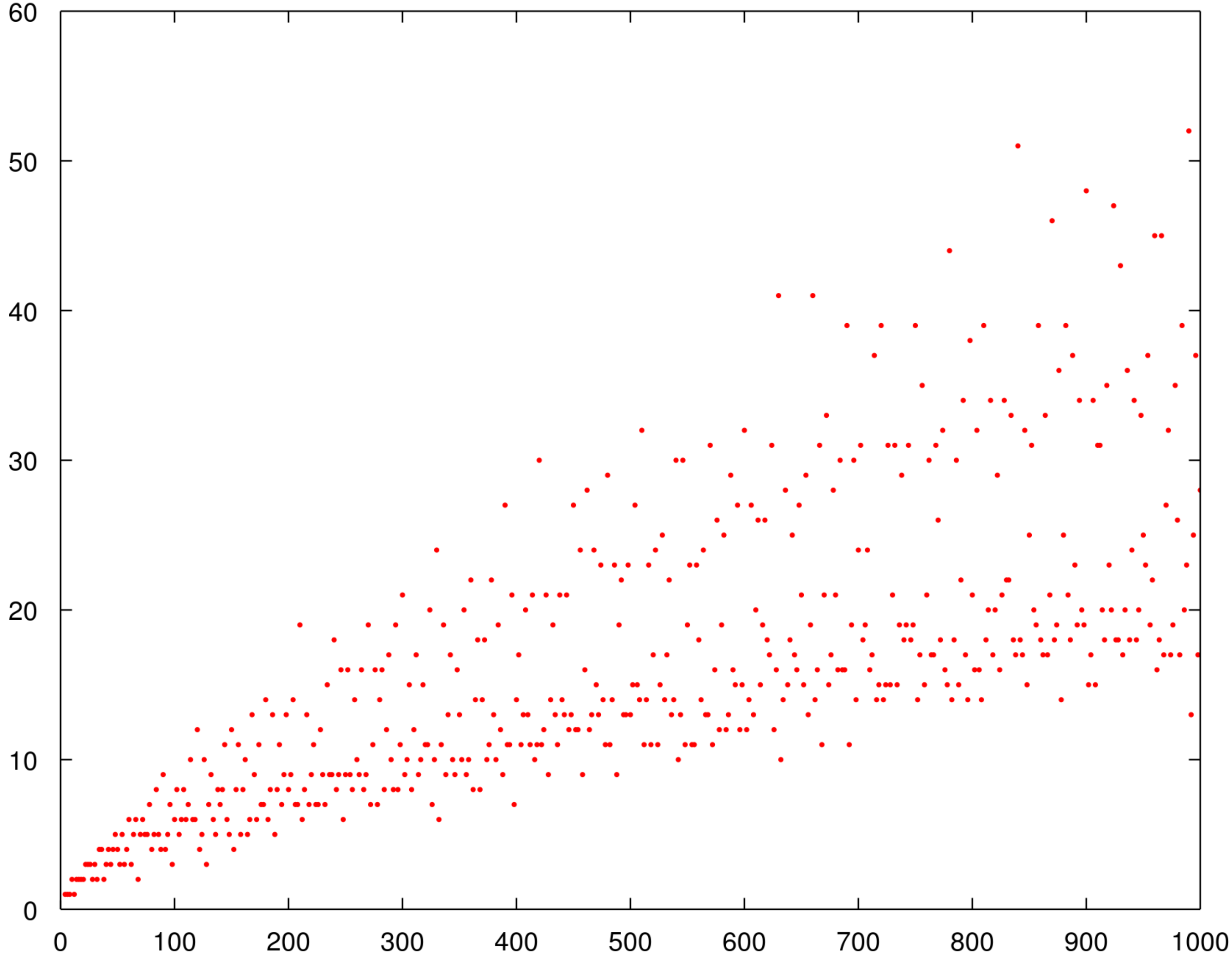
Euer Hochedelgebohrnen ergebenster Diener

Goldbach.

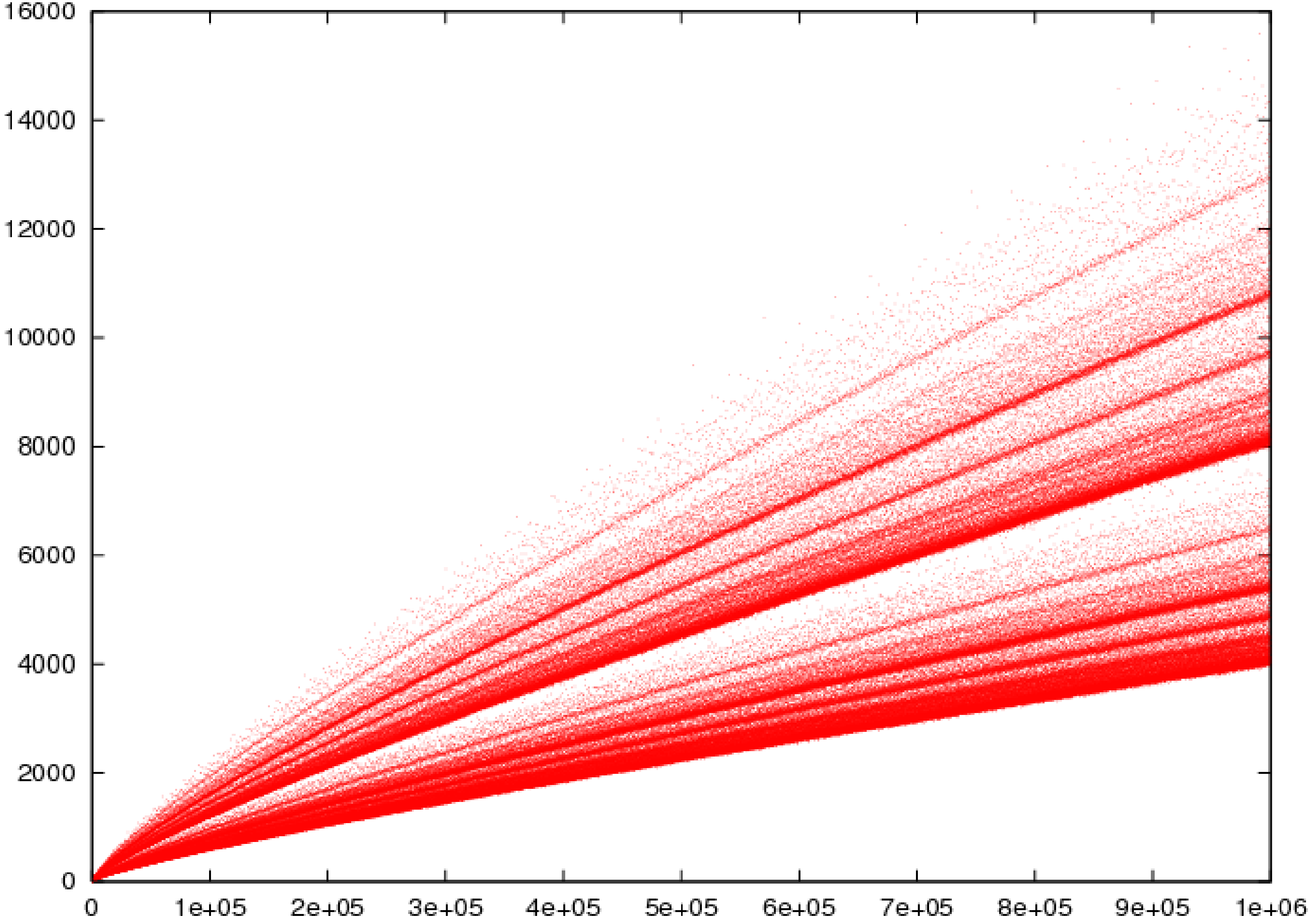
Moscau d. 7. Jun. st. n. 1742.

*Nachdem ich dieses wieder durchgelesen, finde ich, dass sich die conjecture in summo rigore demonstriren lässet in casu $n+1$, si successerit in casu n. et $n+1$ dividi possit in duos numeros primos. Die demonstration ist sehr leicht. Es scheinet wenigstens, dass jede Zahl, die grösser ist als 2, ein aggregatum trium numerorum primorum sey.

G(m) for alle tall < 1000 – varierer mye, men har en litt uklar nedre grense som vi ikke greier å vise er >0



G(n) for alle tall < 1 mill – varierer mye, men har en skarp nedre grense som vi ikke greier å vise er > 0 !



Hvordan vise Goldbach med et program

- En riking i USA har utlovet $1mill for løsning på 10 matematiske problemer som lenge var uløst.
  - Goldbach er en av disse problemene, og premien er ikke hentet.
  - To måter å vise Goldbach:
    - Komme med et matematisk bevis for at den er sann.
    - Komme med et mot-eksempel: Partallet M som ikke lar seg skrive som summen av to primtall.

Hvordan løser vi dette ? Første sekvensielt

- Hvilket program skal vi lage?
  a) **Prog1**: Som for alle tall m=6,8,….,n finner minst én slik sum m= p1+p2
  b) **Prog2**: Som regner ut G(m) for m=6,8,….,n
  c) **Prog3**: Som tester noen tall > 4×10^{18} om Goldbach gjelder også for disse tallene
  (sette ny rekord – ikke testet tidligere, kanskje finne ett tall som ikke lar seg skrive som summen av to primtall.)

Forskjellen på Prog1 og Prog3 er at i Prog1 antar vi at vi vet alle primtall < m, det trenger vi ikke i Prog3 (som da blir en god del langsommere).

Prog1 og Prog2 bruker 'bare' Eratosthenes Sil fra Oblig2.

Prog3 bruker både Eratosthenes Sil og faktoriseringa fra Oblig2.

Skisse av Prog1 : Finn minst én slik sum m=p1+p2, ∀ m <n, m partall, p1 ≤ p2; sekvensielt program

```
<Les inn: n>
<Lag e= Eratosthens Sil(n)>

for (int m = 6 ; m < n; m+=2) {
   // for neste tall m, prøv alle primtall p1 ≤ m/2
    for (int p1 = 3, p1 <= m/2; e.nextPrime(p1){
        if ( e.isPrime(m-p1)) {
           //  System.out.println( m+" = "+ p1 + " + "+ (m-p1) );
           break;
        }
    } // end p1
     if (p1 > m/2) System.out.println(
                  " REGNEFEIL: (Goldbach bevist for n < 4*10**18:"
                   + m + " kan ikke skrives som summen av to primtall ");
} // end m
```

Skisse av Prog2 : Finn G(m) antall slike summmer: m = p1+p2, ∀ m <n, m partall, p1 ≤ p2; sekvensielt program

```
<Les inn: n>
<Lag e= Eratosthens Sil(n)>
int Gm;

for (int m = 6 ; m < n; m+=2) {
     Gm = 0;
    // for neste tall m, prøv alle primtall p1 ≤ m/2
     for (int p1 = 3, p1 <= m/2; e.nextPrime(p1){
         if ( e.isPrime(m-p1)) { Gm++; }
     } // end p1
     if (Gm == 0)
            println(" REGNEFEIL: (Goldbach bevist for n < 4*10**18):"
                       + m + " kan ikke skrives som summen av to primtall ");
      else println(" Antall Goldbachsummer i "+m+" er:"+Gm);
} // end m
```

Betraktninger før skissen til Prog3

- Det største antall bit vi kan ha i en bit-array = maksimalstørrelsen av en int (– litt for overflyt)
- max int = 2 147 483 647
- Prøver å lage en Erotosthanes Sil med n= 214748000
  - Da kan vi faktoriser tall < 4 611 670 350 400 000 002
  - Dette gir oss plass for å prøve ut 611 670 350 400 000 000 tall > $4*10^{18}$
- Egentlig skal vi jo prøve ut alle p1 < m/2, men forskningen viser: (One record from this search: 3 325 581 707 333 960 528 needs minimal prime 9781 (= den største p1 : n =p1+p2))
  - Dvs, satser på at: Er det en Goldbach sum for 19-sifrete tall, så er p1 < 2 147 483 647 (som er mye større enn 9781)

Skisse av Prog3 : Finn minst én slik sum m=p1+p2, sekvensielt program ∀ m, $4*10^{18} < m < 4*10^{18}$ + antall, m partall, p1 ≤ p2;

```
<Les inn: antall>
<Lag e= Eratosthens Sil(2147480000)> // nær øvre grense for int

for (long m = 4*10^18 ; m < 4*10^18+antall; m+=2) {
    // for neste tall m, prøv alle primtall p1 ≤ m/2
     for (int p1 = 3, p1 <= m/2; e.nextPrime(p1){
         if ( e.faktorisering (m-p1).size() == 1 )) {
             //  Funnet Goldbach sum
             break;
         }
     } // end p1
      if (p1 > m/2) System.out.println( " BINGO: Funnet $1. mill:"
                       + m + " kan ikke skrives som summen av to primtall ");
} // end m
```

Prog1: Hvordan parallellisere å finne én sum ?

- Vi har en dobbelt løkke (m og p1)
- Parallelliser den innerste løkka:
  - Deler opp primtallene og lar ulike tråder summere med ulike p1
  - Sannsynligvis uklokt?
- Parallelliserer den ytterste løkka:
  - Hvis vi lar hver tråd få 1/k-del av tallene å finne p1 for.
  - Med n= 2mrd. får hver tråd 250 mill. tall å sjekke.

```
<Les inn: n>
<Lag e= Eratosthens Sil(n)>

for (int m = 6 ; m < n; m+=2) {
   // for neste tall m, prøv alle primtall p1 ≤ m/2
    for (int p1 = 3, p1 <= m/2; e.nextPrime(p1){
       if ( e.isPrime(m-p1)) {
            break;  // funnet sum
       }
    } // end p1
     if (p1 > m/2) System.out.println(" REGNEFEIL for "+
m);
} // end m
```

  - Litt ulik belastning på trådene, da større tall må lete lenger etter den første p1 hvor m-p1 er primtall.
- Parallelliserer ved å lage en rekursjon på ytterste løkke.
  - Kan bruke PRP
  - Mye kortere kode

Konklusjon: Forsøker først å parallellisere den ytterste løkka ?

Viktig poeng: Hvilke rutiner bruker vi?

- Vi har sekvensielle og parallelle versjoner av Eratosthenes Sil og Faktorisering. Hvilken versjon bruker vi i Prog1,2,3?

```
<Les inn: antall>  /*  Prog3 –skisse */
<Lag e= Eratosthens Sil(2147480000)> // nær øvre grense for int

for (long m = 4*10^18 ; m < 4*10^18+antall; m+=2) {
   // for neste tall m, prøv alle primtall p1 ≤ m/2
    for (int p1 = 3, p1 <= m/2; e.nextPrime(p1){
        if ( e.faktorisering (m-p1).size() == 1 )) {
              //  Funnet Goldbach sum
              break;
        }
    } // end p1
     if (p1 > m/2) System.out.println( " BINGO: Funnet $1. mill:"
                    + m + " kan ikke skrives som summen av to primtall ");
} // end m
```

- Kan vi parallellisere inne i en parallellisering ?
- Bør vi evt. gjøre det – og hvorfor / hvorfor ikke ?

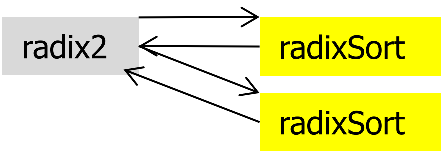
Prog 1,

finn én sum:	n	sekv(ms)	para(ms)	Speedup	SekElem(us)	ParaElem(us)
EratosthesSil	2000000000	14872.89	7386.36	2.01		
Goldbach Sum	2000000000	131447.23	31222.86	4.21	0.06572	0.01561
Total tid	2000000000	146350.58	38813.12	3.77		
EratosthesSil	200000000	1156.82	336.60	3.44		
Goldbach Sum	200000000	11224.35	2646.68	4.24	0.05612	0.01323
Total tid	200000000	12395.19	2990.45	4.14		
EratosthesSil	20000000	69.13	29.94	2.31		
Goldbach Sum	20000000	946.12	252.25	3.75	0.04731	0.01261
Total tid	20000000	1015.25	273.34	3.71		
EratosthesSil	2000000	6.26	1.92	3.27		
Goldbach Sum	2000000	81.11	18.89	4.29	0.04056	0.00945
Total tid	2000000	87.38	20.61	4.24		
EratosthesSil	200000	0.56	0.40	1.38		
Goldbach Sum	200000	6.84	2.90	2.36	0.03421	0.01452
Total tid	200000	7.42	3.30	2.25		
EratosthesSil	20000	0.06	0.43	0.13		
Goldbach Sum	20000	0.84	1.73	0.48	0.04187	0.08665
Total tid	20000	0.89	2.04	0.44		
EratosthesSil	2000	0.01	0.36	0.03		
Goldbach Sum	2000	0.35	1.81	0.19	0.17411	0.90483
Total tid	2000	0.36	2.17	0.17		

Oblig 3

- Parallelliser Radix-sortering med to sifre
- Skriv rapport om speedup for n= 1000, 10 000, 100 000, 1 mill., 10 mill og 100 mill.
- Radix består av to metoder, begge skal parallelliseres.
- Den første, finn max(a[]) – løst tidligere
- Den andre har tre steg – løses i parallell effektivt:
  - a) tell hvor mange det er av hvert sifferverdi i a[] i count[]
  - b) legg sammen verdiene i count[] til ‘pekere til b[]
  - c) flytt tallene fra a[] til b[]
- Steg a) er løst i ukeoppgave (hvor bla. hver tråd har sin kopi av count[])

Den første av to algoritmer som 2-siffer Radix består av.



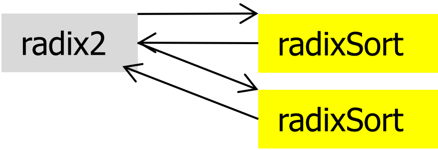

```
static void radix2(int [] a) {
      // 2 digit radixSort: a[]
      int max = a[0], numBit = 2, n =a.length;

     // finn max verdi i a[]
      for (int i = 1 ; i <= n ; i++)
           if (a[i] > max) max = a[i];

      while (max >= (1<<numBit) )numBit++; // antall siffer i max

      // bestem antall bit i siffer1 og siffer2
      int bit1 = numBit/2,
          bit2 = numBit-bit1;

      int[] b = new int [n];
      radixSort( a,b, bit1, 0);    // første siffer fra a[] til b[]
      radixSort( b,a, bit2, bit1);// andre siffer, tilbake fra b[] til a[]
 }
```




```java
/** Sort a[] on one digit ; number of bits = maskLen, shiftet up 'shift' bits */
static void radixSort ( int [] a, int [] b, int maskLen, int shift){
      int  acumVal = 0, j, n = a.length;
      int mask = (1<<maskLen) -1;
      int [] count = new int [mask+1];

     // a) count=the frequency of each radix value in a
      for (int i = 0; i < n; i++)
         count[(a[i]>> shift) & mask]++;

     // b) Add up in 'count' - accumulated values
      for (int i = 0; i <= mask; i++) {
           j = count[i];
            count[i] = acumVal;
            acumVal += j;
       }

     // c) move numbers in sorted order a to b
      for (int i = 0; i < n; i++)
         b[count[(a[i]>>shift) & mask]++] = a[i];

}// end radixSort
```

Hva så vi på i Uke11

I) En del sluttkommentarer og optimalisering av Oblig2

II) Debugging av parallelle programmer

III) Et større problem – uløst problem siden 1742

- Vil du tjene 1 millioner $?
- Løs Goldbach's påstand (alternativt: motbevis den)
- Formulering og skisse av løsning av tre av Goldbachs problemer (Prog1, Prog2, Prog3)
- Parallellisering av disse (ukeoppgave neste uke)

IV) Oblig 3