INF2440 – Prøveeksamen, løsningsforslag, 20 mai 2015

Arne Maus

PSE,

Inst. for informatikk

Prøveeksamen

Er en modell av hva du får til eksamen:

- like mange (+-1) oppgaver som eksamen og nesten samme type oppgave på samme sted.
- Prøveeksamen er grovt sett de oppgavene i to oppgavesett som ikke var gode nok til å komme med til eksamen.
- Dette oppgavesettet har 160 poeng totalt – dvs 4*60min/160= 1 min og 30 sek. per. poeng - dvs. 15 min på oppgave 1

- Flere trykkfeil i prøveeksamen enn i eksamens-settet:
  - Navnet på kurset: Effektiv parallellprogrammering
    ikke: Praktisk parallell programmering
  - tidspunktene gale: Utdeling 11.00 og gjennomgang 15.15.
    Skal være som hjemmesida: Utdeling 10.15 og gjennomgang : 14.15
  - En av oppgavene (3c) står det **kan puffes**. Det betyr at du 'bare' får E på dette punktet hvis det er blankt (ikke F)

Appendix: Modell2 –kode skisse

```
import java.util.concurrent.*;
class Problem {
    //  felles data og metoder A
     public static void main(String [] args) {
          Problem p = new Problem();
          p.utfoer(Runtime.getRuntime().availableProcessors());
     }
     void utfoer (int antT) {
          Thread [] t = new Thread [antT];
          for (int i =0; i< antT; i++)
              ( t[i] = new Thread(new Arbeider(i))).start();

           for (int i =0; i< antT; i++) t[i].join();
     }

     class Arbeider implements Runnable {
       int ind; // lokale data og metoder B

           Arbeider (int in) {ind = in;}
           public void run( ) {
                  // kalles når tråden er startet
           } // end run
     } // end indre klasse Arbeider
} // end class Problem
```

Husk **ikke** skriv **all** kode

N.B. Du skal ikke skrive hele programmet, men bare den sekvensielle metoden, de datastrukturer du trenger og den/de metodene du trenger i det parallelle tilfellet som kalles fra run() - metoden. For datadeklarasjonene, skriv med kommentar i koden hvilke områder (A eller B) i vedlegget du tenker disse plassert. Du kan eventuelt lage en ny konstruktør i class Arbeider hvis du trenger det.

Oppgave 1 (10 poeng)

Når main-tråden pluss de k trådene vi har i et parallelt program alle synkroniserer med en **CyclicBarrier a = new CyclicBarrier(k**), beskriv kortfattet ulike varianter av hva som da kan skje (hvem venter, hvem fortsetter).

Svar: De k første trådene som greier å slippe samtidig igjennom barrieren, den ene som kommer sist blir hengende å vente (og programmet vil ikke terminere).

Etter all sannsynligvis vil main-tråden slippe gjennom barrieren, og det er én av de nystartede trådene som henger – feil uansett.

Dvs. gal kode – skulle vel vært **a = new CyclicBarrier(k+1**);
eller at main-tråden ikke deltok i synkroniseringen.

Oppgave 2 (15 poeng)

Forklar kortfattet de tre kravene ('lovene') vi har når vi har felles, globale variable (eks **int a,b;**) som minst to tråder ønsker å lese og/eller skrive på disse variablene. Når er det sikkert å lese, og når er det sikkert å skrive på disse variablene fra trådene som kjører i parallell?

Uke 4:

A) Hvis ingen skriver på en felles variabel, kan alle lese samtidig uten synkronisering.
B) Hvis to tråder vil skrive på en felles variabel, må slik skriving synkroniseres (f.eks med en synchronized metode i et felles objekt eller en felles ReentrantLock)
C) Hvis én tråd skriver på en felles variabel må ingen andre tråder lese denne variabelen

Regel B) kan brytes hvis det er 'likegyldig' hvilken av trådene som får skrive og ingen feil oppstår hvis noen leser 'gammel' eller annen verdi.

Oppgave 3 (25 poeng)

Effektivisér og parallelliser summering av alle elementer i **`int[][] a.`**

```
long tid = System.nanoTime();
long sum =0;
for (int i = 0; i < n; i++){
      long s =0;
      for (int j = 0 ; j < n; j++){
            s += a[j][i];
       }
       sum +=s;
 }
double seqTid =  (System.nanoTime() - tid)/ 1000000.0
System.out .println ("Sum av a["+n+"]["+n+"]="+sum+
                     ", paa:"+seqTid +" msek");
```

To ineffektiviter:

a) Arrayen a[][] leses kolonne-vis – ikke radvis (følg cache-linjene)

IKKE: s += a[j][i]; **men:** s += a[i][j];

b) Løkker som er inne i 'lang kode bør pakkes inn i små metoder (JIT – komp.)

Løsningsskisse oppg. 3 - sekvensiell

Mange mulige løsninger for å lage metode av deler av koden, denne kan jeg nytte på i sekvensiell og parallell løsning.

```
        // sum langs radene
        for (int i = 0; i<n; i++){
              sum += radSum(a[i]);
         }
.....................

        long radSum (int [] a ) {
                        long sum =0;
                        for (int j = 0 ; j < a.length; j++) {
                                sum += a[j];
                         }
                        return sum;
              } // end radSum
```

Speedup ≥ 100 sammenlignet med første kode (N=25 000)

Oppg. 3 parallell

(speedup ca. 210, n=25 000, k=8)

A

```
int numThreads =
          Runtime.getRuntime().availableProcessors();
 Thread t []= new Thread [numThreads];
 long[] svar = new long [numThreads];
 long  sum =0;
 CyclicBarrier synk= new  CyclicBarrier(numThreads) ;
……..

       for (int tr = 0; tr < numThreads; tr++)
             t[tr] = new Thread(new Para(tr));

          tid = System.nanoTime();

           for (int i=0; i< numThreads; i++)
               t[i] . start();

          try{
                for (int i = 0; i< numThreads; i++)
                     t[i].join();
           } catch(Exception e) {};
```

B

```
class Para implements Runnable{
     int ind;
     int left, right, part;

      Para(int i, int met ) {
           ind =i;
           part = n/numThreads;
           left = ind*part;
           right = left+part ;
           if (ind == numThreads-1) right = n;
       }

  public void run() {
     long s = 0;
     for (int i=left; i < right; i++) {
          s+= radSum(a[i]);
      }
     svar[ind] = s;

      try{ synk.await();}
       catch (Exception e) { return;}

      if (ind == 0) {
         for (int i = 0; i < numThreads; i++) {
                sum+= svar[i];
    } } } // end run
```

```
ArraySum numThreads:8

Inline i loop, kolonnevis sum= 7812542677079, n= 25000, paa:  32349.967 msek
Egen metode, radvis           , n=25000, speedUp:  104.73
Para: Egen Metode,radvis      , n=25000, paa:  158.18 msek, Speedup: 204.51
```

Oppg. 4 – hvor lange ord bruker Ibsen og hvor ofte?

1) Lag en sekvensiell metode som leser **`String [] ordene`** og som lager en tabell over:

den %-vise delen av ordbruken som hadde lengde = 1 (ordene ‘å’ og ‘i’)
den %- vise delen av ordbruken som hadde lengde = 2
.....
den %- vise delen av ordbruken som hadde lengde = 20.

N.B. Hver bruk av et ord teller da med, slik at i f.eks. setningen: «Å å å så glad jeg skal bli.» har tre av 8 ord lengde = 1, ett ord har lengde = 2, to har lengde =3 og to har lengde =4, og dette skal oppgis slik at 37,5% har lengde =1,...osv.

2) Skriv en parallellversjon av løsningen.

Oppgave 4 sekvensiell

```
class IbsenFrekvens{
        int numThreads;
A       final static int MAX_LEN = 21;
        int [] antall = new int[MAX_LEN];
        int[][] traadSvar;
        int [] parAntall= new int[MAX_LEN];
        CyclicBarrier synk = new CyclicBarrier(numThreads);
        String[]ordene ={"En","stue","på","Østråt",
                                "Gennem","den", "åbne", ...}
        int n = ordene.length;
    ..............
      void lagAnalyse (int [] tab, int left, int right){
              for (int i = left ; i < right; i++){
                    tab[ordene[i].length()]++;
       } }
     ..............
        long tid = System.nanoTime();
           lagAnalyse (antall,0,ordene.length);
        tid =  System.nanoTime() - tid;
```

```
for (int j = 1; j < MAX_LEN; j++)    System.out.println("Ord med len = «
    +Format.align(j,2)+" er: "+ Format.align((antall[j]*100.0/n),6,2)+"% av alle ord");
```

Oppgave 4 parallell

B

```
class Para extends Thread {
        int ind;
        int left, right, part;
        int localTab[] = new int[MAX_LEN];

public void run() {
            lagAnalyse (localTab, left, right);
            traadSvar[ind] = localTab;
            try{  synk.await();  }catch (Exception e) { return;}

            if (ind == 0) {
                  for (int i = 0; i < numThreads; i++) {
                        int ant =0;
                        for (int j = 0; j < MAX_LEN; j++){
                              ant += traadSvar[i][j];
                        }
                  parAntall[i] = ant;                                    }
              } // ind == 0
  } // end run
```

```
 Sekvensiell: antall ord:64,
paa:      0.012 msek
Ord med len =  1 er:   3.12% av alle ord
Ord med len =  2 er:  17.19% av alle ord
Ord med len =  3 er:  26.56% av alle ord
Ord med len =  4 er:   7.81% av alle ord
Ord med len =  5 er:  17.19% av alle ord
Ord med len =  6 er:   4.69% av alle ord
Ord med len =  7 er:   6.25% av alle ord
Ord med len =  8 er:   9.38% av alle ord
Ord med len =  9 er:   1.56% av alle ord
Ord med len = 10 er:   3.12% av alle ord
Ord med len = 11 er:   3.12% av alle ord
Ord med len = 12 er:   0.00% av alle ord
Ord med len = 13 er:   0.00% av alle ord
Ord med len = 14 er:   0.00% av alle ord
Ord med len = 15 er:   0.00% av alle ord
Ord med len = 16 er:   0.00% av alle ord
Ord med len = 17 er:   0.00% av alle ord
Ord med len = 18 er:   0.00% av alle ord
Ord med len = 19 er:   0.00% av alle ord
```

```
 Parallell: antall ord:64,
      paa:      1.163 msek, speedup :0.0105
Ord med len =  1 er:   3.12% av alle ord
Ord med len =  2 er:  17.19% av alle ord
Ord med len =  3 er:  26.56% av alle ord
Ord med len =  4 er:   7.81% av alle ord
Ord med len =  5 er:  17.19% av alle ord
Ord med len =  6 er:   4.69% av alle ord
Ord med len =  7 er:   6.25% av alle ord
Ord med len =  8 er:   9.38% av alle ord
Ord med len =  9 er:   1.56% av alle ord
Ord med len = 10 er:   3.12% av alle ord
Ord med len = 11 er:   3.12% av alle ord
Ord med len = 12 er:   0.00% av alle ord
……..
```

Oppgave 5 – finn de to sorterte radene i a[][] - sekv

```
class FinnToSorterte{
                int numThreads;
                int [][] a;
                int n;
                ReentrantLock lock =
                        new ReentrantLock ();
                int [] answer = new int[2];
                int numFound = 0;
.....

  boolean update (int radNum) {
                lock.lock();  // block until condition holds
                 try { answer[numFound ++]=radNum;
                } finally { lock.unlock(); }
                 return numFound == 2;
    } // end update

    boolean radTest (int [] aa ) {
                for (int j = 1 ; j < a.length; j++){
                        if ( aa[j-1] > aa[j]) return false;
               }
                return true;
     } // end radSum
```

```
.....
long tid = System.nanoTime();
// sum langs rad
  for (int i = 0; i<n; i++){
       if (radTest(a[i]) && update(i)) break;
  }
  tid =  System.nanoTime() - tid;

  double seqTid = (tid/1000000.0);
  System.out .println (" Sekvensiell, n="
     + n +", 1.sort:"+answer[0]+
     ", 2.sort:"+answer[1]+", paa:"+
      Format.align(seqTid, 6,2)+"ms." );
```

Oppg.5 parallell:

```
class Para implements Runnable{
        int ind;
        int left, right, part;

        Para (int i ) {
                ind =i;
                part = n/numThreads;
                left = ind*part;
                right = left+part ;
                if (ind == numThreads-1) right = n;
            }

         public void run() {
              for (int i = left; i<right; i++){
                   if (radTest(a[i]) && update(i) ) break;
               }
        } // end run

    } // end Para
```

```
FinnToSorterte numThreads:8
Sekvensiell, n=20000, 1.sort:13638, 2.sort:16906, paa:  2.26ms.
Parallell,      n=20000, 1.sort:13638, 2.sort:16906, paa:  1.38ms., Speedup:  1.64
```

Oppg 6, Beste k= antall tråder, gitt antall kjerner

```
class BesteAntTraader{
            int [] a;
            int n,k;
            long sum, fasitSum;
            long [] svar ;
           CyclicBarrier synk ;


...
          int kk = Runtime.getRuntime().
                    availableProcessors();
          Thread t []= new Thread [3*kk];
          double [] seqTid = new double[11],
                    paraTid = new double[11];
           svar = new long [3*kk];
```

```
long radSum (int [] a, int l, int r ) {
            long s =0;
            for (int j = l ; j < r; j++)
                 s += a[j];
             return s;
 } // end radSum


....

   long tid = System.nanoTime();
   fasitSum = radSum(a,0,n);
   tid =  System.nanoTime() - tid;
    seqTid [m] = (tid/1000000.0);
```

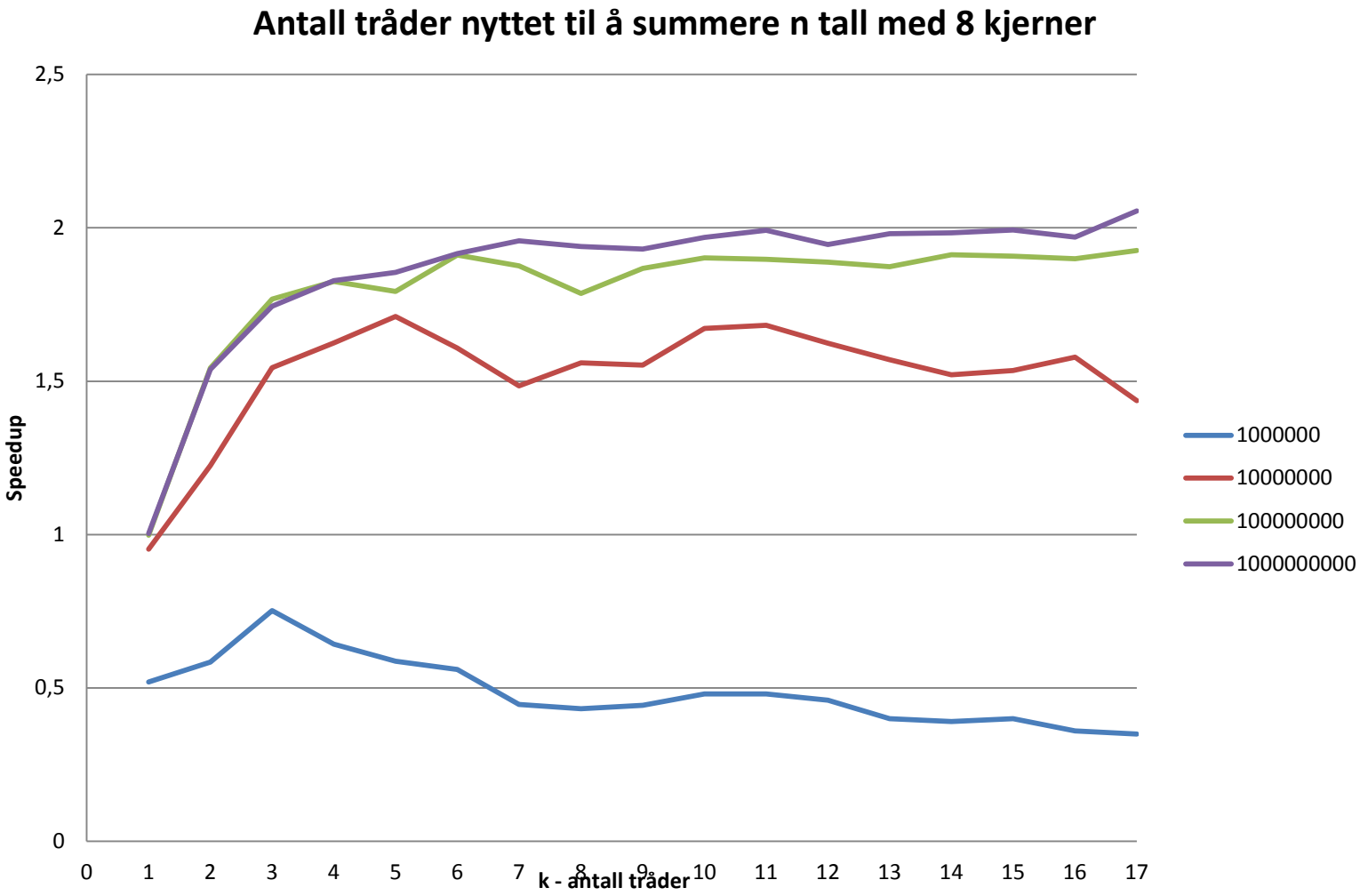
Oppg. 6 parallell

```
for ( k = 1; k < 3*kk; k++) {
            sum =0;
            tid = System.nanoTime();
            for (int tr = 0; tr < k; tr++)
                        (t[tr] = new Thread(new Para(tr))).start();

             try{
                        for (int i = 0; i < k; i++){   t[i].join(); }
            } catch(Exception e) {return;};
             paraTid[m] =  (System.nanoTime() - tid)/1000000.0;
}
...
 public void run() {
        svar[ind] = radSum(a,left,right);
         try{
               synk.await();
          }catch(Exception e) {return;}
          if (ind == 0 ) {
               for ( int i =0; i < k; i++) {
                    sum += svar[i];
           }
       }

   } // end run
```

Oppg6 – resultater med 8 kjerner



Om eksamen

- Kunnskapsspørsmål:oppg.2
- Hvordan virker synkronisering opp 1
- Lag et sekvensielt og parallelt program – oppgavene 3,4,5 og 6

LYKKE TIL !