UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Prøveeksamen i:	INF2440— Praktisk parallell programmering
Prøveeksamensdag:	20. mai 2015
Tidspunkter:	10.15 Utdeling av prøveeksamen
	14:15 Gjennomgang av løsning (i StoreAud, KN hus)
Oppgavesettet er på:	4 sider
Vedlegg:	Skisse av Modell2-koden .
Tillatte hjelpemidler:	Alle trykte og skrevne

- Kontroller at oppgavesettet er komplett, og les nøye gjennom oppgavene før du løser dem. Poengangivelsen øverst i hver oppgave angir maksimalt antall poeng.
- Du kan legge dine egne forutsetninger til grunn og gjøre rimelige antagelser, så lenge de ikke bryter med oppgavens "ånd". Gjør i så fall rede for disse forutsetningene og antagelsene.
- Til eksamen skal svarene skrives på gjennomslagspapir. Da må du huske å skrive hardt nok til at besvarelsen blir mulig å lese på alle gjennomslagsarkene, og ikke legge andre deler av eksamensoppgaven under når du skriver.
- Til eksamen skal du selv beholde underste arket etter levering av de to øverste til eksamensinspektøren. Nummerer sidene, og husk å skrive kandidatnummeret ditt på besvarelsen.
- En av oppgavene (3c) står det **kan puffes**. Det betyr at du 'bare' får E på dette punktet hvis det er blankt (ikke F)

I vedlegget finner du en litt forenklet versjon av Modell2-koden som ble nyttet i kurset (en ytre klasse med main-tråden og en indre klasse hvor objekter av denne blir egne tråder). I den skissen er det markert med store bokstaver ulike områder av denne koden som det blir referert til i oppgavene.

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).

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?

Oppgave 3(25 poeng)

Gitt en todimensjonal matrise **`int[][] a = new int [n][n]`** som er fylt med tilfeldige heltall, og noen har skrevet følgende kode for å summere alle elementene i **`a[][]`** i en **`long sum`**, slik:

```
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");
```

Dette er kode inne i en større løkke hvor bla. n varieres og hvor vi kjører hver n-verdi flere ganger for å få en rimelig riktig tid for en bestemt n.

Oppgave:

a. Dette er for så vidt riktig kode, men med alt du vet om effektiv kode og JIT-kompilering, skal du skrive den om (som sekvensiell) kode slik at den blir betydelig raskere.
   (når n =25000 vil en slik bedre sekvensiell kode kunne gå ca. 100 ganger raskere).
b. Skissert hvordan du kan parallellisere din betydelig bedre sekvensielle kode.
   (når n = 25000 skulle da programmet gå ca. 200 ganger fortere enn koden ovenfor.)

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 4 (30 poeng)

Du har fått lest inn i: **`String[] ordene;`** alle ordene vi finner i skuespillene til Henrik Ibsen; og slik at hvis et ord, som f.eks. 'han', forekommer n ganger i skuspillene, så er det n av arrayelementene i **`ordene[]`** som akkurat inneholder det ordet (her: 'han').

Vi er nå interssert i å undersøke den %-vise lengden av ordene som Ibsen har brukt, og vi skal anta at ingen av ordene er lengre enn 20 bokstaver.

Oppgave:

a) 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.

b) Skriv en parallellversjon av løsningen.

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 data-deklarasjonene, 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 5 (40 poeng)

Skriv først en sekvensiell og så en effektiv parallell metode for å løse følgende problem: I en to-dimensjonal array: `int[][]a = new int[n][n];` er det slik at to av radene er sortert, mens de resterende n-2 radene ikke er sortert (dvs. i hver av disse n-2 radene er det minst en indeks `i` slik at i rad k er: `a[k][i] > a[k][i+1]).` Oppgaven din nå er å finne disse to radene`.` La til slutt main-tråden skrive ut svaret (indeksen på de to radene som er sortert) i både det sekvensielle og parallelle programmet.

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 data-deklarasjonene, 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 6 (40 poeng)

Vi skal her skrive et program som undersøker hvor mange tråder det lønner seg å bruke til å løse det enkle problemet å summere n = 1 million tilfeldige heltall i en array (her er skisse av deler av den sekvensielle koden):

```
int [] a = new int[n];
……  fyll a[], ta tid ved kall på sumArray , print…….

long sumArray (int [] a) {
       long sum =0;

       for (int i = 0; i < a.lemgth; i++) {
             sum += a[i];
       }
       return sum;
} // end sumArray
```

Du skal nå lage et parallelt program og skrive hele koden som tester de tider du får ved å øke k = antall tråder du bruker fra 1 og oppover til 3*antall kjerner på maskinen du kjører på. Den tiden du måler skal inkludere det å lage k-tråder, la de k-trådene summere hver sin del, summere de k delsummene, og så til sist vente i main-tråden (bruk f.eks. join()). Tidene presenteres i millisekunder.

Siden du vet at første gang kjører programmet vil JIT-kompilering gjøre at senere kjøringer går langt raskere. Du skal da for hver verdi av k kjøre testen m = 11 ganger i en løkke og presentere medianen av de 11 kjøringene som den 'riktige' verdien for denne k-verdien. Du kan anta at en metode for innstikksortering er tilgjengelig (**`double innStikkSort (double [] t) {..}`**) uten at du behøver å skrive kode for den.

Oppgave: Skriv dette parallelle programmet, som i tillegg til å presentere medianverdiene for de ulike k-verdiene, skriver ut en konklusjon om hvilken k-verdi som var best.

Appendix – Modell2 kodeskisse:

```
import java.util.concurrent.*;
class Problem {
    //  felles data og metoder A
     public static void main(String [] args) {
           Problem p = new Problem();
           p.utfoer(12);
     }
     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(int ind) {
                   // kalles når tråden er startet
            } // end run
     } // end indre klasse Arbeider
} // end class Problem
```