UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Prøveeksamen i :	INF2440— Praktisk parallell programmering
Prøveeksamensdag :	26. mai 2014
Tidspunkter:	11.00 Utdeling av prøveeksamen
	15:15 Gjennomgang av løsning (i StoreAud, KN hus)
Oppgavesettet er på :	5 sider
Vedlegg :	Skisse av Model2-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 Model2-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)

a) Når vi synkroniserer i Java, hva annet skjer i tillegg til at vi evt. får ordnet trådene tidsmessig (svar kortfattet, maksimalt 10 linjer)

b) Hvilken effekt ville det ha på programutførelsen av et parallelt program i Java hvis alle metodene var `synchronized` og deklarert i A-området (se vedlegget)

Oppgave 2 (10 poeng)

Du skal beskrive forskjellene av Gustafsons lov kontra Amdahls lov. Anta at du har et program som for en gitt n har 10% sekvensiell kode. Hva sier de to lovene om hva som vil skje med hvor stor maksimal speedup vi kan få på dette programmet hvis vi kjører programmet med en langt større n på problemet. Hvilken av de to lovene er mest optimistiske med hensyn til å få størst mulig speedup.

Oppgave 3(25 poeng)

Goldbach hadde egentlig to påstander i 1743. Den første om partall har vi jobbet mye med. Hans andre påstand var at ethvert oddetall kan skrives som summen av tre primtall. Dette er visstnok bevist i 2013 i et 133 siders bevis (sist revidert 14. april 2014!).

a) Skisser hvordan du vil lage et sekvensielt program som sjekker dette for alle oddetall n < M, og hvor M er et vilkårlig stort tall. Du kan godt bruke din løsning på Goldbachs påstand om partall som en del av løsningen.
b) Skisser så hvordan du vil parallellisere det å vise Goldbachs påstand om alle oddetall n < M.
c) **Kan puffes:** Hvis Goldbachs påstand om partall var bevist (noe det ikke er), men anta likevel det. Bevis da at Goldbachs påstand om oddetall er sann (meget lett, tips: 3 er et primtall).

Oppgave 4 (20 poeng)

Anta at at du **har startet 5 tråder og at** du i felt A i vedlegget deklarerer følgende to variable:

```
int teller2 =0
Semaphore abc = new Semaphore(2);
```

Trådene eksekverer kode i sin run()-metode som før eller siden alle prøver å utføre:

```
try{abc.acquire();
catch (Exception e) {return;}
teller2 ++;
```

Vi antar at dette gjør trådene slik at det ikke blir data-kappløp om **`teller2`** mellom trådene.

a) Hva er verdien av **`teller2`** etter at første tråden prøver å utføre denne koden.
b) Hva er verdien av **`teller2`** etter at andre tråden prøver å utføre denne koden.
c) Hva er verdien av **`teller2`** etter at tredje tråden prøver å utføre denne koden.
d) Hva er verdien av **`teller2`** etter at fjerde tråden prøver å utføre denne koden.

Oppgave 5 (40 poeng)

Skriv først en sekvensiell og så en parallell metode for å løse følgende problem: I en matrise: **`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 begge deler, 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 (20 poeng)

Du har et trådbasert system og vil øke en **`int`** variabel **`num`** hver gang med 1 slik at den til slutt blir så stor at den av maskinen blir betraktet som et negativt tall ved at verdien kommer opp i fortegnsbitet til variabelen. For det formålet starter du opp 10 tråder. Du skal da benytte følgende to programsetninger s1 og s2:

```
// setninger s1
int num =1;

// setninger s2
synchronized int leggTil(int num) {
    num++ ;
    return num;
}
```

Vi tenker oss at det i hver run-metode i trådene utføres følgende kode:

```
while (num > 0 ){
  num = leggTil(num);
}
```

Selve problemløsningen din tar utgangspunkt følgende sekvensielle kode:

```
while (num >= 0) num++;
System.out.println (" num < 0: ” + num+ ",  i mål”);
```

Spørsmål: Hva er effekten av å plassere de to setningene (s1 og s2) ulike steder i Model2koden (vedlegget):

a) s1 og s2 i A ?
b) s1 i A og s2 i B?
c) s1 i B og s2 i A?
d) s1 og s2 i B?

Gi en kort begrunnelse på hvert av punktene a)-d).

Oppgave 7 (70 poeng)
Flettesortering, sekvensiell og parallell.

(For dere som har glemt hva flette-sortering er: Du har i en array a[] to sorterte sekvenser s1 og s2 (av lengde l1 og l2) som ligger etter hverandre i a[] – dvs. at hvis s1's siste element er i a[k], så starter s2 i a[k+1]. Kopier disse to over i en array b[] slik at du stadig tar det minste elementet av s1 eller s2 i a[] som enda ikke er kopiert som neste element du kopierer over til b[]. Du plukker altså elementene fra toppen hver av de to sekvensene til alle elementene er kopiert over til b[]. I b[] er det etter kopieringa da en sortert sekvens av lengde = l1+l2.)

a) Du skal nå først skrive en sekvensiell, rekursiv versjon av flettesortering, som består av to metoder (omtrent som quicksort) - her er den første oppgitt:

```
void fletteSort(int [] a) {
        if (a.length <= 50) insertSort(a,0,a.length-1);
        else {
             int [] b = new int [a.length];
             flette(a,b,0,a.length-1);
        }
  } // end fletteSort
```

Du skal skrive den rekursive metoden flette:

```
void flette (int [] a, int [] b, int v, int h){
    < ..din kode her.. >
 }
```

som sorterer en del av en heltallsarray fra og med element a[v] til og med element a[h] – dvs: a[v..h] som følger:

1. Du skal bruke: **`void insertSort (int [] a, int v, int h)`** som sub-algoritme, hvis (h-v) ≤ 50. Da skal du sortere den delen med insertSort.
2. På 'backtrack' (etter retur av de to rekursive kallene i flette), skal du flette de to sorterte delene av a[v..h] ved:
   i. Flette-sortere dem over i 'sin del' av arrayen b – dvs. i b[v..h].
   ii. Kopiere den sorterte (nå om lag dobbelt så lange) sekvensen fra b[] til tilbake til samme plasser i a[].
   iii. Når da det opprinnelige, første kallet på flette returnerer, er hele a[] sortert.

II) Skriv en effektiv parallell versjon av din sekvensielle, rekursive flettesorteringsalgoritme.

N.B. Også her skal du **ikke** skrive hele programmet (du kan referere til koden i vedlegget)), men bare den sekvensielle metoden **`flette`**, og de datastrukturer du trenger og den/de

metodene du trenger i det parallelle tilfellet som kalles fra run() - metoden. For begge deler, 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.

Appendix – Model2 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
```