

# UNIVERSITETET I OSLO

## Det matematisk-naturvitenskapelige fakultet

Eksamen i: INF2440— Praktisk parallell programmering  
Eksamensdag: 2. juni 2014  
Tidspunkter: 14.30 – 18.30  
Oppgavesettet er på: 4 sider  
Vedlegg: Skisse av Modell2-koden  
Tillatte hjelpemidler: Alle trykte og skrevne modell

- 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.

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) Hva er en tråd i Java (svar: Maksimalt 5 linjer)
- b) Hvis vi har  $k$  kjerner i en multikjerne maskin, hvor mange tråder kan vi da ha minimalt og maksimalt i et Java-program?

### Oppgave 2 (10 poeng)

Du skal her se på konsekvenser av Amdahls lov. Anta at du har et program med 25% sekvensiell kode og da de resterende 75% som kan parallelliseres. Anta at du har 8 kjerner og kan parallellisere de 75% perfekt, og at det er ikke noen overhead med å starte og stoppe tråder. Hvilken speedup kan du da forvente deg av ditt 'perfekt' parallellisert program; utled svaret ,ikke bare svar et tall?

### Oppgave 3 (15 poeng)

Anta at du i felt A i vedlegget deklarerer følgende variabel, og at du **har startet 15 tråder**:

```
CyclicBarrier grind = new CyclicBarrier(14);
```

Trådene eksekverer kode i sin run()-metode, som før eller siden alle utfører :

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

- Beskriv situasjonen når den første tråden utfører denne koden.
- Beskriv situasjonen når den 13de tråden utfører denne koden.
- Beskriv situasjonen når den 14de tråden utfører denne koden.
- Beskriv situasjonen når den 15de tråden utfører denne koden.

### Oppgave 4 (20 poeng)

Anta at du har to tråder som aksesserer en felles variabel, deklartert i felt A slik: `int i = 0;`

Tråd0 utfører følgende kode: `i++; i++;` før den terminerer, mens

Tråd1 utfører: `i--; i--;` før den terminerer. Det er ingen synkronisering.

Hvilke mulige verdier er det at `i` har etter at begge trådene har terminert?

Tegn diagrammer med tidsakse som viser *hvordan* hver av de verdiene du hevder kan bli verdien av `i` til sist, kan fremkomme (når leser og skriver T0 og T1 hvilke verdier?):

### Oppgave 5 (50 poeng)

Skriv først en sekvensiell og så en parallell metode for å beregne største kolonne-summen (= summen av elementene i en kolonne) i en matrise: `int a[][] = new int[n][n];` La til slutt main-tråden skrive ut svaret (indeksen til kolonnen med størst sum+ selve summen).

- Før du skriver den parallelle koden skal du beskrive minst to, og helst tre måter å dele opp matrisen for de ulike trådene. Vurder de oppdelingene du beskriver om hvilken av dem du mener vil gi raskest parallell kode, og begrunn kortfattet svaret.
- Skriv parallell kode for den oppdelinga du har valgt.

**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.

## Oppgave 6 (15 poeng)

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

```
int teller1 = 0;
Semaphore abc = new Semaphore(1);
```

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

```
try{ abc.acquire();
    teller1++;
    abc.release();
} catch (Exception e) {};
```

- Hva er verdien av `teller1` rett før den første tråden utfører `abc.release()`;
- Hva er verdien av `teller1` rett før den andre tråden utfører `abc.release()`;
- Hva er verdien av `teller1` rett før den tredje tråden utfører `abc.release()`;
- Hva er verdien av `teller1` rett før den fjerde tråden utfører `abc.release()`;
- Må noen av trådene vente ved utførelsen av punktene a)-d). Begrunn svaret kort.
- Hvis vi like etter koden ovenfor i hver run() metode skriver ut verdien av `teller1` slik:  
`System.out.println("Tråd num: "+ind+", teller1:"+teller1);`  
Er vi da sikre på at de verdiene på vi svarte på `teller1` i pkt. a) – d) vil bli skrevet ut? Begrunn svaret.

## Oppgave 7 (70 poeng)

Du har en stor mengde små heltall i byte-arrayen `a[]` (f.eks flere hundre millioner), og en søkestring i en byte-array `s[]` (f.eks 4-10 tall) og er interessert å finne ut følgende to ting:

- Hvor forekommer søkestringen `s[]` i `a[]`. Svaret er en **ArrayList eksaktLikhet** med startindeksen i `a[]` til alle de stedene hvor det er en slik eksakt likhet.
- Hvor er det en sekvens av tall i `a[]` som forekommer `s[]`, men hvor det er eksakt en feil. Dvs. at det finnes en sekvens av fortløpende tall i `a[]` som er lik søketallene i `s[]`, men hvor vi har at eksakt en av tallene i `a[]` ikke er lik motsvarende tall i `s[]`. – svaret er her også en **ArrayList medEnFeil** med startindeksene i `a[]` til alle stedene hvor det er en slik type likhet.

Husk at hvis vi søker etter `s[] = 123123` og `a[]` inneholder `...0012312312342399999` ..., så vil vi finne `123123` to ganger med null feil og en gang med 1 feil i dette området av `a[]`. Vi aksepterer altså at to slike funn overlapper hverandre, men det er jo helt avhengig av søkenøkkelen om det er aktuelt.

(dette tilsvarer på en måte søking etter et gen i en DNA-streng, og søking etter et gen med en feil i en DNA-streng).

Hvis du deler opp data i `a[]` på en eller annen måte i parallellisering av a) og b) må du ta

hensyn til at en slik likhet som vi leter etter kan krysse en slik delelinje.

- I) Det er relativt enkelt å se at man kan skrive (både den sekvensielle og parallelle metoden) slik at vi bare via en parameter kan avgjøre om vi i et kall på metoden skal finne 0 eller 1 feil . Vurder kortfattet hvor fornuftig det er tidsmessig med en slik felles metode er i det tilfellet hvor vi skal finne 0 feil (perfekt likhet).
- II) Skriv et sekvensiell og et parallelt program av disse to relativt like søkeprogrammene.

**N.B.** Også her skal du **ikke** skrive hele programmet (du kan referere til koden i vedlegget), 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.

## 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(8);
    }
    void utfoer (int antT) {
        Thread [] t = new Thread [antT];
        for (int i =0; i< antT; i++)
            ( t[i] = new Thread(new Arbeider(i))).start();
        try{
            for (int i =0; i< antT; i++) t[i].join();
        } catch(Exception e) {}
    }
}

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
```