UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	INF3380 — Parallellprogrammering for naturvitenskapelige problemer
Eksamensdag:	6. juni 2014
Tid for eksamen:	9.00 – 13.00
Oppgavesettet er på 4 sider.	
Vedlegg:	Ingen
Tillatte hjelpemidler:	Kalkulator Ett to-sidig A4 ark med håndskrevne notater

Kontroller at oppgavesettet er komplett før
du begynner å besvare spørsmålene.

Vekting av oppgavene

Oppgave 1: 10%
Oppgave 2: 15%
Oppgave 3: 15%
Oppgave 4: 35%
Oppgave 5: 25%

Oppgave 1 (vekt 10%)

Anta at et beregningsproblem er blitt dekomponert i mange små oppgaver, slik at antall oppgaver er større enn antall tilgjengelige prosessorer. Hva er de to hovedprinsippene for en god fordeling av disse oppgavene blant prosessorne?

Oppgave 2 (vekt 15%)

2a (vekt 5%)

Anta at p prosessorer danner en 1D "linear array". Forklar hvorfor følgende modell for tidsbruk er riktig for "broadcast" av en melding av lengden m fra en prosessor til alle andre prosessorer:

$$T_P = (t_s + t_w m)\,\lceil \log_2 p \rceil,$$

hvor t_s og t_w er to konstanter, mens $\lceil \cdot \rceil$ betegner den såkalte "ceiling"-funksjonen.

(Fortsettes på side 2.)

2b (vekt 5%)

Dersom det er totalt $p = 12$ prosessorer i den 1D lineære arrayen, forklar steg for steg hvordan datatransport skjer mellom prosessorne.

2c (vekt 5%)

La oss nå se på en 2D mesh av prosessorer, hvor antall rader er r og antall søyler er s. Utledd tidsbruk-modellen for dette tilfellet.

Oppgave 3 (vekt 15%)

Anta at omgivelsesvariabelen `OMP_NUM_THREADS` er satt til å være 4. Hva vil følgende OpenMP kodesnutt generere som resultat (skrevet ut av `printf`)? Begrunn svaret ditt.

```
  int *result_array = (int*)malloc(100*sizeof(int));
  int i, num_threads;

#pragma omp parallel default(shared)
  {
    int thread_id = omp_get_thread_num();

    if (thread_id==0)
      num_threads = omp_get_num_threads();

#pragma omp for schedule(static,2)
    for (i=1; i<=20; i++)
      result_array[thread_id] += i;

  }

  for (i=0; i<num_threads; i++)
    printf("Result from thread %d is %d\n",i,result_array[i]);
```

Oppgave 4 (vekt 35%)

Gitt en vektet graf med A som betegner dens $n \times n$ "adjacency matrix". Floyds algoritme løser "all-pairs shortest paths"-problemet på følgende måte:

1. **procedure** FLOYD_ALL_PAIRS_SP(A)
2. **begin**
3. $D^{(0)} = A$;
4. **for** $k := 1$ **to** n **do**
5. **for** $i := 1$ **to** n **do**

(Fortsettes på side 3.)

```
6.                          for j := 1 to n do
7.                              d_{i,j}^{(k)} := min(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)});
8.          end FLOYD_ALL_PAIRS_SP
```

4a (vekt 10%)

Vi antar at radene til den $n \times n$ "adjacency matrix" A er fordelt på p prosessorer ifølge en 1D blokk-partisjonering. Beskriv i detaljer kommunikasjonene som trengs i en MPI parallellisering av Floyds algoritme. (Du trenger ikke å implementere en hel MPI kode.)

4b (vekt 15%)

Utledd modellen for tidsbruk $T_P(n, p)$ av MPI-parallelliseringen som er basert på en 1D blokk-partisjonering av radene til "adjacency matrix" A.

4c (vekt 10%)

Utledd den assosierte funksjonen av "isoefficiency".

Oppgave 5 (vekt 25%)

"Gauss elimination"-algoritmen er som følgende (hvor A betegner en $n \times n$ matrise, b og y betegner to vektorer av lengden n):

```
1.          procedure GAUSSIAN_ELIMINATION (A, b, y)
2.          begin
3.              for k := 0 to n − 1 do              /* Outer loop */
4.              begin
5.                  for j := k + 1 to n − 1 do
6.                      A[k, j] := A[k, j]/A[k, k];   /* Division step */
7.                  y[k] := b[k]/A[k, k];
8.                  A[k, k] := 1;
9.                  for i := k + 1 to n − 1 do
10.                 begin
11.                     for j := k + 1 to n − 1 do
12.                         A[i, j] := A[i, j] − A[i, k] × A[k, j]; /* Elimination step */
13.                     b[i] := b[i] − A[i, k] × y[k];
14.                     A[i, k] := 0;
15.                 endfor;             /* Line 9 */
16.             endfor;                 /* Line 3 */
17.         end GAUSSIAN_ELIMINATION
```

(Fortsettes på side 4.)

5a (vekt 10%)

Skriv en sekvensiell C funksjon

```
void gauss_elim(double **A, double *b, double *y)
```

som implementerer algoritmen ovenfor.

5b (vekt 15%)

Lag en OpenMP parallellisering av den C funksjonen ovenfor. Diskuter kvaliteten på OpenMP-parallelliseringen, med hensyn til overhead og lastbalansering.