Suggested solutions for the INF3380 exam of spring 2010

Oppgave 1 (10%)

Du har n oppgaver som alle tar like lang tid, og en datamaskin med P prosessorer. Vi antar at n er større enn P og er ikke delelig på P.

Forklar hvordan du kan fordele oppgavene så jevnt som mulig på prosessorene.

Suggested solution: Let $\mathrm{mod}(\cdot,\cdot)$ denote the modulo function, $\lceil\cdot\rceil$ the ceiling funciton, and $\lfloor\cdot\rfloor$ the floor funciton. Then, $\mathrm{mod}(n,P)$ processors will get $\lceil\frac{n}{P}\rceil$ tasks each, and the remaining processors will each get $\lfloor\frac{n}{P}\rfloor$ tasks.

Oppgave 2 (10%)

Du har n oppgaver som alle tar en "tidsenhet" å gjennomføre. Du har i tillegg n oppgaver som tar dobbelt så lang tid, d.v.s. to tidsenheter.

Forklar hvordan du kan fordele de $2n$ oppgavene best mulig på P prosessorer. (Her antar vi at n er større enn $2P$, og n går ikke opp i P.)

Suggested solution: First, we divide all the "heavy" tasks among the P processors as above. The result is that $P-\mathrm{mod}(n,P)$ processors have only got $\lfloor\frac{n}{P}\rfloor$ "heavy" tasks each. These processors will then each be added with two "light" tasks, so that all the processors have now received identical work load. Finally, the remaining "light" tasks, $n-2(P-\mathrm{mod}(n,P))$ in total, will be divided as evenly as possible among all the P processors.

Oppgave 3 (10%)

Vi har et endimensjonalt array i C av lengde n. Vi ønsker å dele dette opp i P deler, med så jevn størrelse som mulig. (n er større enn P og går ikke opp i P.)

Du skal implementere følgende funksjon i C:

```
void divide(int n, int P, int k, int* start_k, int* stop_k)
```

hvor `n,P,k` er input til funksjonen og `k` er et heltall mellom 0 og $P-1$. Funksjonen skal returnere verdier for `start_k` og `stop_k`, som skal være startindeksen og endeindeksen for del nummer `k` av arrayet.

Suggested solution:

```
void divide(int n, int P, int k, int* start_k, int* stop_k)
{
  *start_k = (n*k)/P;
  *stop_k = (n*(k+1))/P-1;
}
```

Oppgave 4 (20%)

Vi har følgende C funksjon:

```
double Euclidean_norm (double* a, int n)
{
    double sum = 0.;
    int i;
    for (i=0; i<n; i++)
       sum += a[i]*a[i];

    return sqrt(sum);
}
```

Lag en parallell versjon av denne funksjonen med MPI. (Hint: funksjonen `divide` fra Oppgave 3 kan brukes her.)

Suggested solution:

```
double Euclidean_norm_MPI (double* a, int n)
{
  double sum, local_sum = 0.;
  int i, i_start, i_stop;
  int my_rank, total_procs;

  MPI_Comm_Rank (MPI_COMM_WORLD, &my_rank);
  MPI_Comm_Size (MPI_COMM_WORLD, &total_procs);

  divide(n, total_procs, my_rank, &i_start, &i_stop);

  for (i=i_start; i<=i_stop; i++)
    local_sum += a[i]*a[i];

  MPI_Allreduce (&local_sum, &sum, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

  return sqrt(sum);
}
```

Oppgave 5 (20%)

Lag en parallell versjon av `Euclidean_norm` med OpenMP.

Suggested solution:

```
double Euclidean_norm_OMP (double* a, int n)
{
  double sum = 0.;
  int i;
#pragma omp parallel for shared(a) reduction(+: sum)
  for (i=0; i<n; i++)
    sum += a[i]*a[i];

  return sqrt(sum);
}
```

Oppgave 6 (10%)

Hva menes med ”deadlock”? Gi et eksempel.

Suggested solution: In the context of MPI communication, a deadlock arises if a communication operation forever remains uncompleted. An example can be as follows:

```
if (my_rank==0) {
  MPI_Recv (&a, 1, MPI_INT, 1, tag1, MPI_COMM_WORLD, &status);
  MPI_Send (&b, 1, MPI_INT, 1, tag2, MPI_COMM_WPRLD);
}
else if (my_rank==1) {
  MPI_Recv (&a, 1, MPI_INT, 0, tag2, MPI_COMM_WORLD, &status);
  MPI_Send (&b, 1, MPI_INT, 0, tag1, MPI_COMM_WPRLD);
}
```

Oppgave 7a (10%)

Utled Karp-Flatt metrikken (**ikke lenger pensum for 2013**).

Suggested solution: Suppose the parallel time usage has three parts:

$$T(n,p) = \sigma(n) + \varphi(n)/p + \kappa(n,p),$$

where n denotes the problem size, p denotes the number of processors used, $\sigma(n)$ denotes the inherently serial component of the computation, φ denotes the portion of computation that is parallelizable, and κ denotes the parallel overhead.

We also note that $T(n,1) = \sigma(n) + \varphi(n)$, that is, parallel overhead does not apply in the serial time usage.

Now, let us define $e(n,p)$ as

$$e(n,p) = \frac{(p-1)\sigma(n) + p\,\kappa(n,p)}{(p-1)T(n,1)} = \frac{\sigma(n) + \frac{p}{p-1}\kappa(n,p)}{T(n,1)}.$$

Therefore, following the above definition of $e(n,p)$, we will have

$$e(n,p)T(n,1) \;=\; \sigma(n) + \frac{p}{p-1}\kappa(n,p).$$

Moreover, we have

$$\begin{aligned}\frac{1-e(n,p)}{p}T(n,1) &= \frac{1}{p}\left(T(n,1)-e(n,p)T(n,1)\right)\\ &= \frac{1}{p}\left(\sigma(n)+\varphi(n)-\sigma(n)-\frac{p}{p-1}\kappa(n,p)\right)\\ &= \frac{\varphi(n)}{p}-\frac{1}{p-1}\kappa(n,p)\,.\end{aligned}$$

Consequently, combining the above two results, we will get

$$\begin{aligned}e(n,p)T(n,1)+\frac{1-e(n,p)}{p}T(n,1) &= \sigma(n)+\frac{p}{p-1}\kappa(n,p)+\frac{\varphi(n)}{p}-\frac{1}{p-1}\kappa(n,p)\\ &= \sigma(n)+\kappa(n,p)+\frac{\varphi(n)}{p}\\ &= T(n,p)\,.\end{aligned}$$

At the same time, the definition of speedup Ψ can be used to give the following relation between Ψ and $e(n,p)$:

$$\begin{aligned}\Psi(n,p) &= \frac{T(n,1)}{T(n,p)}\\ &= \frac{T(n,1)}{\left(e(n,p)+\frac{1-e(n,p)}{p}\right)T(n,1)}\\ &= \frac{1}{e(n,p)+\frac{1-e(n,p)}{p}}.\end{aligned}$$

When the value of Ψ is known and the above equation is solved with respect to e, we will recover exactly the Karp-Flatt metric:

$$e=\frac{1/\Psi-1/p}{1-1/p}.$$

Oppgave 7b (10%)

Se på denne tabellen over målt tidsbruk: Hvordan kan Karp-Flatt hjelpe deg å forstå mulige årsaker til inperfekt skalering?

P	1	2	3	4	5	6	7	8
Tidsbruk	10	5.35	3.83	3.10	2.68	2.42	2.24	2.12

Suggested solution: Using the Karp-Flatt metric, we can compute e as follows:

P	1	2	3	4	5	6	7	8
Tidsbruk	10	5.35	3.83	3.10	2.68	2.42	2.24	2.12
Ψ	N/A	1.87	2.61	3.23	3.73	4.14	4.46	4.71
e	N/A	0.070	0.075	0.080	0.085	0.090	0.095	0.1

That is, the experimentally determined serial fraction e increases with P, meaning that the reason for the poor speedup is due to parallel overhead.