SOLUTIONS TO INF3380 PROBLEMS, WEEK 3

Exercise 1

It makes sense to initially distribute $\lfloor \frac{n}{P} \rfloor$ tasks to each process.

(Here $\lfloor \cdot \rfloor$ simply means "round down.")

If we are lucky, this accounts for all the tasks, but in general we are left with a remainder term r that is somewhere between 0 and $P-1$. In code, this is simply

```
int init_dist = n / P;
int remainder = n % P;
```

[1]

If we label the processes by $p_0, p_1, ..., p_P$, it is logical to distribute the r remaining tasks among the r first processes, p_0, p_1, ..., p_{r-1}. We can easily implement this in code:

```
int my_tasksize, tasksize_total, myrank, numprocs;

// Find out myrank, numprocs, tasksize_total here
// ...

my_tasksize = tasksize_total / numprocs;
if (myrank < tasksize_total % numprocs) my_tasksize ++;
```

Exercise 2

Directly:

	Worker 0	Worker 1	Worker 2	Worker 3
$t = 0$ min	Begin T0	(idle)	(idle)	(idle)
$t = 10$ min	T0 complete; begin T1	Begin T2	(idle)	(idle)
$t = 20$ min	T1 complete; begin T3	(etc.) Begin T4	Begin T5	Begin T6
$t = 30$min	Begin T7	Begin T8	Begin T9	Begin T10
$t = 40$min	Begin T11	Begin T12	Begin T13	Begin T14
$t = 50$min	T11 complete	T12 complete	T13 complete	T14 complete

Note that there is nothing to be gained from increasing the number of workers to $5, 6$ or 7.

[1]You should be familiar with the remainder operator %. It is frequently referred to as the modulo operator, which is a great name to use if you want to make it really unclear what it does.

A similar table for 3 workers:

	Worker 0	Worker 1	Worker 2
$t = 0$ min	Begin T0	(idle)	(idle)
$t = 10$ min	T0 complete; begin T1	Begin T2	(idle)
$t = 20$ min	T1 complete; begin T3	(etc.) Begin T4	Begin T5
$t = 30$ min	Begin T6	Begin T7	Begin T8
$t = 40$ min	Begin T9	Begin T10	Begin T11
$t = 50$ min	Begin T12	Begin T13	Begin T14
$t = 60$ min	Complete	Complete	Complete

Exercise 3

We'll label the processes $p_{i,j}$, where $0 \le i \le P-1$, $0 \le j \le Q-1$. We can split the problem into two one-dimensional problems of the exercise 1 variety, that is, we demand that each process will have a workload of size $k_i \times l_j$. Then

$$k_i = \left\lfloor \frac{m}{P} \right\rfloor + e_i \,,$$
$$l_j = \left\lfloor \frac{n}{Q} \right\rfloor + f_j \,,$$

where

$$e_i = \begin{cases} 1 & \text{if } i < m \bmod P \,, \\ 0 & \text{otherwise.} \end{cases}$$
$$f_j = \begin{cases} 1 & \text{if } j < n \bmod Q \,, \\ 0 & \text{otherwise.} \end{cases}$$

(This is a nice demonstration of how there's no problem simple enough that you can't make it indecipherable with math.)

Again translated into C code:

```
int k, l, m, n, mycoords[2], numprocs_cartesian[2];

// Find out mycoords, numprocs_cartesian, m, n here
// ...

k = m / numprocs_cartesian[0];
if (mycoords[0] < m % numprocs_cartesian[0]) k ++;

l = n / numprocs_cartesian[1];
if (mycoords[1] < n % numprocs_cartesian[1]) l ++;
```

Remark: Obviously, this distribution will usually be strictly less fair than the 1d variant, and it is possible to make more fair distributions if you're willing to divide work with crazier shapes.

More stuff: There is a nice structure to this distribution. Specifically, given any process $p_{i,j}$, its workload is of the same height as all its left-right neighbours; similarly it has the same width as all its above-below neighbours. Okay, so this is a rather trite observation, but it does make for easy communication between neighbouring processes.

Exercise 4

Put $t_{i,j}$ to be subtask j of task i, $0 \leq i \leq n-1$, $0 \leq j \leq m-1$; for reference, the pipeline has the following structure.

	Stage 0	Stage 1	(...)	Stage $m-1$
$t=0$	Begin $t_{0,0}$		(...)	
$t=1$	Begin $t_{0,1}$	Begin $t_{1,0}$	(...)	
(...)	(...)			
$t=m-1$	Begin $t_{m-1,0}$	Begin $t_{m-2,1}$	(...)	Begin $t_{0,m-1}$

As the name suggests, the setup is shaped rather like a pipe, where tasks flow continuously through the pipe from left to right. Subtask $t_{i,j}$ can only commence when the preceding subtasks $t_{i,0}$, $t_{i,1}$, $\cdots$, t_{i-j-1} have completed; once a subtask has passed through a stage, the stage is ready to commence work on a new subtask of the same type.

Clearly an entire task takes m time units to pass through the entire m-stage pipe, and the final tasks enters the pipe (i.e. reaches stage 0) at time $t = n-1$. Summing up, the final task will commence at time $t = n-1$, all tasks have passed the pipeline at time $T_{\text{pipe}} := m + n - 1$.

(This is of course an extremely simple model. It is possible that some subtasks take more time to complete, or you can dedicate multiple cores to some or all stages.)

Exercise 5

Computing the tasks sequentially takes time $T_{\text{seq}} := m \cdot n$, so we have

$$p = \frac{T_{\text{seq}}}{T_{\text{pipe}}} = \frac{m \cdot n}{m + n - 1}\,.$$

Now it's just a matter of solving for n,

$$\begin{aligned} mp + np - p &= mn\,, \\ p(m-1) &= n(m-p)\,, \\ n &= p\,\frac{m-1}{m-p}\,. \end{aligned}$$

In particular $p = 1$ implies $n = 1$, (that is, we get no speedup from sending a single task through the pipeline, which makes sense) and $p \to m$ implies $n \to \infty$.