Student ID: .....

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2014
TRIMESTER 2

NWEN303

Concurrent Programming

Time Allowed: TWO HOURS

Instructions: Closed Book.
There are 60 possible marks on the exam.

You are supposed to write as much as a normal one hour exam, however, you have two hours so that you can spend more time to reason, to plan how to proper answer and better choose how to phrase your answers. Please, write clearly.

Answer all questions in the boxes provided.
Every box requires an answer.
If additional space is required you may use a separate answer booklet.

No calculators permitted.
Non-electronic Foreign language dictionaries are allowed.

No reference material is allowed.

Question	Topic	Marks
1.	Relevance of parallel algorithms	28
2.	Designing/implementing parallel algorithms in Java	32
	Total	60

Question 1. Relevance of parallel algorithms [28 marks]

After graduation you started working as a freelance consultant. Many companies are now asking you to understand if their next projects should try to take advantage of multicore machines capabilities or not.

(a) [6 marks] Identify at least 2 benefits of parallel programming. Justify your answer.

Efficiency: using multiple cores we can do more operations for unit of time

Responsiveness: using multiple workers, even if one worker is busy with a certain part of logic, another worker can provide feedback to the user.

(b) [8 marks] Identify at least 4 risks/costs involved in parallel programming. Justify your answer.

Hight developing cost: Doing a parallel program is harder and more time consuming that doing a corresponding sequential one.

Decrease in bug free confidence: Testing a sequential program usually guarantee that, at least for that particular input, the program is behaving as expected. This assumption is shattered by parallel programs.

Difficulties to find competent enough programmers: In the general case, doing a correct parallel program require such a high level of programming skill that employing the right person can be challenging.

Less predictable performance: On different machines, parallel programs can perform in very different ways, and sometimes on the "right" hardware configuration they are even outperformed by sequential ones.

(c) [7 marks] Briefly, using 3 or 4 sentences, describe a problem that would be significantly more efficient when implemented using a parallel programming approach instead of a sequential one. Justify your answer.

Genetic algorithms requires to create/evolve populations of individual and to measure their fitness.
The fitness function is potentially very time consuming, but its only input is the specific individual.
Thus, multiple fitness functions can be executed in parallel, speeding up the overall execution time.
Another example is Video encoding or data compression, This holds since most compression algorithms are limited by CPU.

(d) [7 marks] Briefly, using 3 or 4 sentences, describe a problem that would not have significant benefits when implemented using a parallel programming approach. Justify your answer.

Many interactive programs, spend most of their times simply waiting for input in order to provide some answer to the user.
There is no point in improving efficiency of such programs.
In general, profiling can help identifying the bottleneck of an application. If the bottleneck is not a algorithmic part working on large amount of loosely connected data, usually other forms of optimization would have more success than parallelism.
Another typical example could be managing the download of a file, since the mail bottleneck is not the cpu, a parallel downloaded would not be expected to perform any better than a non-parallel one.

Question 2. Designing/implementing parallel algorithms in Java [32 marks]

PerfectImage is company producing a image processing tool, they have many filters, all implementing the filter interface:

```
1 public interface Filter{
2   void applyFilter(Image img);
3   }
```

Filters takes images and do some in place transformation on the image. This means that `applyFilter` modifies the (large) `Image` object in order to obtain a filtered version of the same image. Is it possible to set up a filter chain in this simple way:

```
1 void applyFilters(ArrayList<Filter>filters,Image img){
2   for(Filter f:filters){
3     f.applyFilter(img);
4   }
5 }
```

Now PerfectImage wants to support movies. In this simple context movies are just collections of images.

A naive attempt of applying all the filters to all the images works correctly, but is too slow.

```
1 void applyAllFilters0(ArrayList<Filter>filters,ArrayList<Image> movie)
2   for(final Image i:movie){
3     for(final Filter f:filters){
4       f.applyFilter(i);
5     }
6   }
7 }
```

They asked to their programmer Bob to write a parallel version for such task. This is the first attempt that Bob comes out with:

```
1  private static final ExecutorService pool=
2      Executors.newCachedThreadPool();
3  void applyFilters1(ArrayList<Filter>filters,ArrayList<Image> movie)
4      throws InterruptedException, ExecutionException{
5    for(final Image i:movie){
6      for(final Filter f:filters){
7        pool.submit(new Callable<Void>() {
8          public Void call() throws Exception {
9            f.applyFilter(i);
10           return null;
11       }});
12     }
13   }
14 }
```

(a) [6 marks] With great happiness of Bob, this method executes very fast! However, when Bob run the test suite on this method, sometimes the test fails stating that some filters was not applied at all.
Describe what happens in this code, and explain where is the error.

Bob is submitting tasks, but is not waiting for them to complete. That is why the method executes very fast, is not waiting for any computation to be performed at all.

After understanding his mistake, Bob modify his code as follows. The only code difference is in the lines marked with //#

```
1  private static final ExecutorService pool=
2      Executors.newCachedThreadPool();
3  void applyFilters2(ArrayList<Filter>filters,ArrayList<Image> movie)
4      throws InterruptedException, ExecutionException{
5    for(final Image i:movie){
6      ArrayList<Future<?>> results=new ArrayList<Future<?>>();//#
7      for(final Filter f:filters){
8        results.add(pool.submit(new Callable<Void>() {//#
9          public Void call() throws Exception {
10           f.applyFilter(i);
11           return null;
12       }}));
13     }
14     for(Future<?> r:results){r.get();}//#
15   }
16 }
```

(b) [6 marks] Now, when Bob, runs the tests, he discovers that some images are corrupted, as if two filters was acting on a given image at the same time. Describe what happens this time:

- Explain why some images are corrupted.
- State the name for this kind of problems.

In this code, for any specific image, all the filters results are submitted and executed, and the execution waits until all the filters are applied before moving to the next image.
There is nothing preventing multiple filters to all act at on the same image at the same time, so the image may end up corrupted.
This is often called race condition

Depressed but still not ready to surrender, Bob learn about the synchronized statement on a web forum and naively tries to use it. The only difference is in the line marked with //#

```
1  private static final ExecutorService pool=
2      Executors.newCachedThreadPool();
3  void applyFilters3(ArrayList<Filter>filters,ArrayList<Image> movie)
4      throws InterruptedException, ExecutionException{
5    for(final Image i:movie){
6      ArrayList<Future<?>> results=new ArrayList<Future<?>>();
7      for(final Filter f:filters){
8        results.add(pool.submit(new Callable<Void>() {
9          public Void call() throws Exception {
10           synchronized(i){f.applyFilter(i);}//#
11           return null;
12       }}));
13     }
14     for(Future<?> r:results){r.get();}
15   }
16 }
```

(c) [8 marks] Now, when Bob, runs the tests, they pass just fine! Bob is very happy and goes to sleep. However, the day after while doing some performance test he discovers that his code was much slower than the sequential version, and only one core is ever used.

Moreover, sometimes the result is still different from what is expected, as if the filters were executing one at the time, but in the wrong order.

Explain why:

- Only one core is used.
- It is even slower than the sequential version.
- The filters can be executed in any order.

In this code, as before, for any specific image, all the filters results are submitted and executed, and the execution waits until all the filters are applied before moving to the next image. This means that all the parallel operations are acting on the same image. by synchronizing on the current image, Bob is forcing all the filters to act one at a time.
This is slower than the sequential version, since there is the overhead of callables, future and executor services.
The filters can still be executed in any order on a given image, since there is no guarantee that any specific order will be followed by executing the various tasks.
Moreover, Bob should probably fix his tests so that they fail if the order of filer application is not the expected one.

(d) [6 marks] At this point Bob gives up and quit, promising that will never use parallelism ever again. Now write your own version that is both correct and efficient. You solution must use futures.

```
private static final ExecutorService pool=
      Executors.newCachedThreadPool();
//here we assume every Image in the movie to be a different object
void applyFilters(ArrayList<Filter>filters,ArrayList<Image> movie)
    throws InterruptedException, ExecutionException{
  for(final Filter f:filters){//just swap lines
    ArrayList<Future<?>> results=new ArrayList<Future<?>>();
      for(final Image i:movie){
        results.add(pool.submit(new Callable<Void>() {
          public Void call() throws Exception {
            f.applyFilter(i);
            return null;
        }}));
    }
    for(Future<?> r:results){r.get();}
  }
}
```

As you can see, is just swapping two lines. This implies a change of mentality: from doing one image at a time, and trying using all the filters, to using a filter at a time, and trying to use all the images. Another, more efficient possibility is to move the filter loop inside the future:

```
private static final ExecutorService pool=
      Executors.newCachedThreadPool();
//here we assume every Image in the movie to be a different object
void applyFilters(ArrayList<Filter>filters,ArrayList<Image> movie)
    throws InterruptedException, ExecutionException{
    ArrayList<Future<?>> results=new ArrayList<Future<?>>();
    for(final Image i:movie){
      results.add(pool.submit(new Callable<Void>() {
        public Void call() throws Exception {
          for(final Filter f:filters){
            f.applyFilter(i);}
          return null;
      }}));
    }
    for(Future<?> r:results){r.get();}
}
```

(e) [6 marks] Now, write a version using the producer consumer pattern.

```
private static final ExecutorService pool=
    Executors.newCachedThreadPool();
void applyF(List<Filter>filters,List<Image> movie)
   throws InterruptedException, ExecutionException{
  BlockingQueue<Image> lastOutputQueue
    =new LinkedBlockingQueue<Image>(movie);
  List<Future<?>> tasks=new ArrayList<Future<?>>();
  for(final Filter f:filters){
    final BlockingQueue<Image> inputQueue=lastOutputQueue;
    final LinkedBlockingQueue<Image> outputQueue
      =new LinkedBlockingQueue<Image>();
    tasks.add(pool.submit(new Callable<Void>(){
      public Void call() throws Exception {
        while(true){
          Image next=inputQueue.take();
         f.applyFilter(next);
          outputQueue.add(next);
        }}}));
    lastOutputQueue=outputQueue;
  }//end for filters
  for(int i=0;i<movie.size();i++){lastOutputQueue.take();}//wait end!
  for(Future<?> f:tasks){f.cancel(true);}//stop the others
  }
}
```

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

* * * * * * * * * * * * * * *

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.