Test 1: Lectures 1 to 13

EEE4084F 2017-04-25

Instructions:

- Answer on separate pages.
- Make sure that your student number is on all your answer pages.
- There are **4** questions, each divided into sub-questions. Answer all questions.
- Total time: 45 minutes.
- Total marks: 50.

Question 1: Converting serial to parallel **[10 Total]**

You are tasked to implement a multi-threaded moving average filter. The filter operation works on separate input (X) and output (Y) buffers each of size N. The moving average has a window size of M (i.e. M is the number of elements that are averaged to get Y_i except at the start and end of the array). The OCTAVE code for this moving average filter is provided below. As you can see the behaviour is a bit different at the start and at the end of the array (i.e. where the indexing would have gone out of bounds).

```
# X : input array 1..N of floats Y : output array 1..N of floats
# M : int window size
function [Y] = maf (X,M)
  N = length(X);
  halfM = round(M/2);
  Y=zeros(1,N);
  for i=1:N
    if (i>halfM)
        if (i<N-halfM)
            Y(i)=mean(X(i-halfM:i+halfM));
        else
            Y(i)=mean(X(i:N));
        endif
    else
        Y(i)=mean(X(1:i));
    endif
   endfor
endfunction
```

You can assume that the size of N will be greater than M (generally M won't be more than about 10). A scenario of the results would be as follows if N was 10 and M was 2, Y(1) = X(1), Y(2) = mean(X(1),X(2),X(3)), Y(3) = mean(X(2),X(3),X(4)) … etc.

Answer the questions below:

#1 What level of granularity would you say this problem is? Is it course gained or fine grained? How much so? Motivate your answer. **[2]**

#2 This code needs to be parallelized using pthreads and should run on a multicore CPU (e.g. an Intel i7). Explain how you would design a solution for this, what threads would you have, what operations would they do. Use an illustration to help your explanation and to show how you would partition the data, and possibly having different tasks. NB: Show any data structures that you would need to define which may be passed to the thread function.

[5]

#3 With possible help from the cheatsheet on the last page, provide a C / C++ implementation of a thread routine for this program (you may or may not decide to have different thread routines, provide a comment as to which routine you are providing). You do not need to show main function code for creating threads. Assume mean is already implemented, as shown on the last page. **[3]**

Question 2: Landscape of Parallel Computing [10 Total]

This question relates to paper 1 titled "The Landscape of Parallel Computing Research: A View from Berkeley" by Asanovic et al.

#1 There is much talk in the paper about old conventional wisdoms and new conventional wisdoms. What is meant by old and new conventional wisdoms? (No need to provide an example for this question).

[2]

#2 Name two old conventional wisdoms and how these have been replaced, in the new 'landscape of computing' by a corresponding new conventional wisdom.

[4]

#3 There is a lot of mention about kernels in terms of parallel computing nowadays, but Asanovic et. al discuss DWARFS. Surely a DWARF is like any other kernel?? Or is it? Explain the difference between the Asanovic et. al's concept of a DWARF and how this may be different from the concept of a kernel in relation to OpenCL or CUDA.

[4]

Question 3: Parallel Computing and Benchmarking [10 Total]

These questions are based on the lectures.

#1 What is meant by the term 'golden measure' in the context of parallel computing? (as in its used in our pracs and lectures.)

[1]

#2 Amdahl's law is sometimes used as a defacto excuse for a parallel solution not giving the awesome speedups as might have been anticipated at the start of a project. What is Amdahl's law? What aspect of it would account for a parallel solution not being so

awesomely fast as hoped? Provide the Amdahl's Law maximum speedup calculation where *f* is the fraction of parallel code and *n* is the number of processors.

[6]

#3 In the lectures the concept of a benchmark suite was mentioned a few times. What exactly is a benchmark suite as opposed to just a benchmark? What does a benchmark suite comprise and why is it better than just a single benchmark (e.g. doing a matrix multiply to decide which platform performs best).

[3]

Question 4: Processing considerations **[20 Total]**

#1 FLOPS and GFLOPS are useful measures, but typically a machine is not doing only floating point operations. Have a look back at the moving average filter in Q1. You can see there aren't really many floating point operations except for ones that are presumably in the function 'mean'. If the function mean was implemented as below:

```
float mean (vector<float> X) {
  float sum=0.0F;
  for (int i=0; i<X.length(); i++) sum+=X[i];
  return (float)sum/X.length();
}
```

Explain how you could get an approximate FLOPS measure for the moving average filter program (remember M can change). You can assume that you've already converted the OCTAVE code into efficient C code. (Bear in mind that even though the machine might be able to do a sustained 10 GFLOPS, e.g. one FP op after another, we would like to figure out how many FLOPS it is doing).

[5]

The following scenario applies to #2 to #3 below…

A sequential process, which operates on 10M 32-bit samples takes 3ms to complete. This speed was too slow for the application, which required the process to complete within 1.5ms. Accordingly, the engineering team went about converting to a parallel pipelined version. They redesigned the system to have a 4-stage pipeline, and after thorough testing determined the average run time for each stage and communication transfer time between stages. Each stage works on a block of 1M samples at a time, so to process the whole 10M input 10 x 1M blocks are fed into the pipeline. Their analysis results are shown in the table below (for stages working at max data block inputs of 1M). The data transfer between stages is done by the source stage and can be done concurrently while the next stage is busy, i.e. if stage *i* needs to transfers to stage *i+1* then it doesn't need to wait for stage *i+1* to complete; however it is designed so that the processing of a new input block does not start until its next stage has completed.

Answer the questions that follow…

Stage → Timing ↓	**1: Remove outliers**	**2: Normalize**	**3: FFT**	**4: Find max**
Computation Time	30 us	50 us	100 us	10 us
Data transferred to next step / final output	1 us	2 us	20 us	1 us

Note this is a software system, not a hardware solution that is clocked. A stage blocks if the next stage is still busy. When the pipeline is filling up the first stage can complete in

#2 How long does it take for the first 1M block to get through the pipeline (i.e. for it to go from stage 1 and out stage 4, assuming that blocks keep getting fed into the system). Show your working. You can draw a table or pipeline sketch to work it out.

[5]

#3 How long does it take to process the whole 10M input? Have the engineers achieved their objective of completing the processing within 1.5ms? What is the speedup of the processing of the new parallel version over the previous sequential version (based on processing 10M of samples, you also do not need to account for any start-up delays or time involved in doing the sampling done, again you can use and show your tables or sketches to work out the answer).

[8]

#4 Nice easy question to end off with. Assume that OCTAVE can run threads which can have parameter: data send to thread, threadid=number of this thread (from 1 to nthreads) and nthreads is the number of threads. What data domain decomposition is used in the code snippet below? You can choose between: a) continuous, b) blocked, c) interlaced, d) interleaved / cyclic.

[2]

```
function thread_ileve (A,threadid,nthreads)
  global global_sum;
  sum = 0;
  for i=threadid:nthreads:length(A)
     sum = sum + A(i);
  end
  global_sum(threadid) = sum;
endfunction
```

END OF TEST

CHEET SHEET

int pthread_create(pthread_t *tid, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg); // attr can be set to NULL for default attributes

void pthread_join(pthread_t thread_id, void **value_ptr);

void pthread_exit(void *value);

int pthread_kill(pthread_t thread_id, int sig);

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

int pthread_mutex_destroy(pthread_mutex_t *mutex);

Question 1 #3:

Implementation of mean that you can assume exists:

```
float mean (float* X, int starti, int endi) {
  float sum=0.0F;
  for (int i=starti; i<=endi; i++) sum+=X[i];
  return (float)sum/X.length();
}
```