UNIVERSITY OF CAPE TOWN

EEE4120F - High Performance Digital Embedded Systems

Class Test 1 [Syllabus: Lectures 1 - 8] - Sample Solutions

20^{th} March 2019

Instructions:

- **Time:** 90 mins
- **Marks:** 66
- Answer each question on separate pages
- Make sure your student number is on all your answer pages
- There are 4 questions, each divided into sub-questions. Answer all sub-questions.

QUESTION ONE [15 MARKS]

1. Consider the "distributed embedded smart camera" system by *Michael Bramberger et al* that we discussed in the lectures. Discuss briefly why this system can be considered an example of a high performance digital embedded computing system. **[3]**

   Ans:

   -parallel computing hardware: hardware architecture is based on a multiprocessor configuration comprising of up to 10 digital signal processors with aggregate performance of to 80GIPS

   -real-time processing: designed for real-time processing e.g real-time video analyses and compression

   -SWaP constraints: designed to meet strict size, weight and power consumption constraints

2. During the lectures, we reiterated the "brick wall" as one of the main design challenges that forced the computing industry to seriously consider parallel computing as a key strategy for achieving high performance in applications. Briefly describe what you understand this "brick wall" to be. **[3]**

   Ans:

   -Brick wall = power wall + memory wall + instruction-level parallelism wall

   -power wall: inability to stack more transistors into one die to produce processors with higher clock cycles due to increase in dynamic power dissipation (i.e $P\alpha\frac{1}{2}.C.V.F^2$*)*

   -memory wall: lower memory access rates on uni-processor architectures create a performance bottleneck

   -instruction-level parallelism wall: limit in ability to extract more instruction-level concurrency from sequential applications

3. Filtering is one of the most fundamental and commonly used operation in digital systems. In particular, the finite impulse response (FIR) filter is a popular example. Given a set of N real filter coefficients, $h = \{h_0, h_1, ..., h_{N-1}\}$, which specifies the desired behavior of the filter, and real input speech samples buffer of size $Z > N$, $x = \{x_0, x_1, ..., x_{Z-1}\}$, the filter calculates each filtered output sample $y(t)$ at time $t = \{t_0, t_1, ..., t_T\}$, by performing a sum of products according to the following formula:

$$y(n) = \sum_{k=0}^{N-1} h(k) * x(n-k).$$

Derive the computational complexity of the sequential FIR filter algorithm. **[7]**

Ans:

-Computational complexity refers to the amount of processing (in terms of operations) required by the algorithm

-For each y(n): N real multiplications + N-1 real additions -> 2N-1 real operations (see formula)

-For all y(n) output samples: given Z input samples -> Z ouput samples -> Z(2N-1) real operations*

-Therefore, if Z>N, which should be the case in practice -> using Big-Oh notation, FIR is of algorithm complexity > $O(N^2)$

4. Assume that you have Y cores on a multi-core processor to run an optimally multi-threaded version of the FIR filter, express the expected speedup factor. **[2]**

Ans:

-Maximum achievable speedup S*, for any algorithm with a parallelisable fraction* p *that can be accelerated with a factor* s*, is provided by Amdahl's law:* $S = \frac{1}{(1-p)+\frac{p}{s}}$

-For the FIR, it is possible to achieve a parallel fraction of p=1 *by partitioning the input samples* X *evenly across the parallel processors*

-However, due to communication overhead, the achievable speedup s *will not equal the number of parallel processors i.e* s<Y

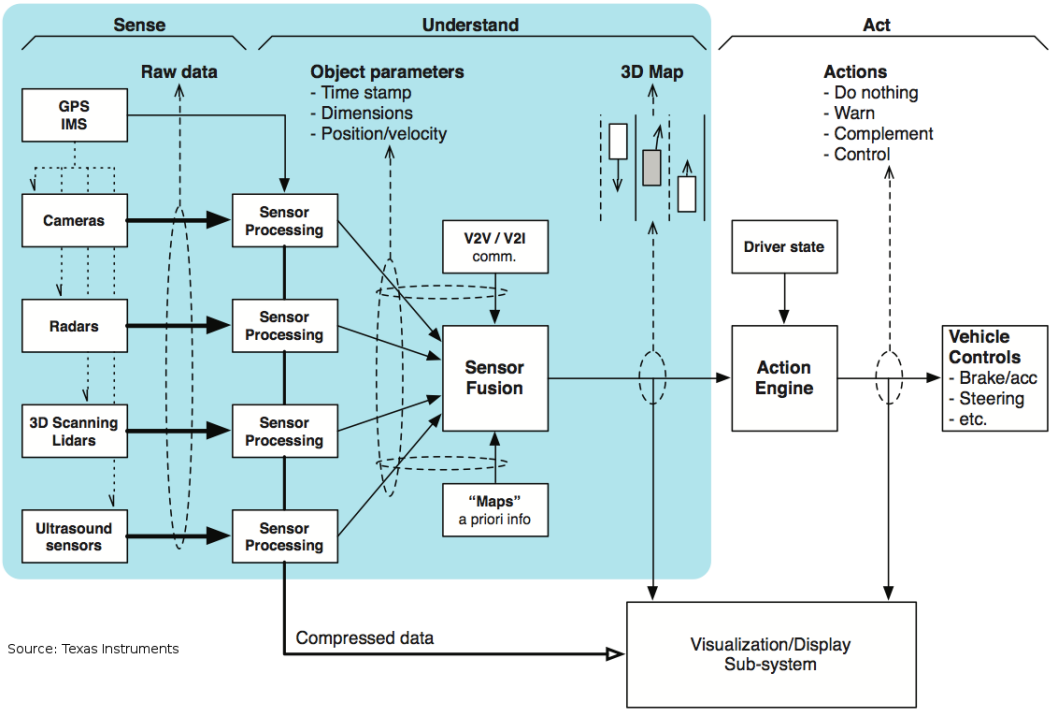
-Therefore: $S = \frac{1}{(1-p)+\frac{p}{s}} = S = \frac{1}{(1-1)+\frac{1}{s}} = s < Y$

QUESTION TWO [23 MARKS]

You will recall that, in the lectures, we considered the technologies behind autonomous cars as one example of practical applications of high performance digital embedded computing systems. The figure below provides a functional view of the flow of data in a typical sensing and control system for an autonomous vehicle. Note that each of the various sensors has a dedicated processing stage that pre-processes raw sensor data and prepares it for usage in creating a real-time object representation that can be used by the next processing stages.

The system uses the outputs of all the sensors to generate a 3D map of the environment surrounding the vehicle. The map shows, among several things, pedestrains, traffic lights, curbs and lane markers, vehicles, the car's location in a larger map of the area and other items that must be

noticed for safe driving. The "action engine" uses this information to determine what action the car should take and communicates activatioin signals to the processors that control the car's mechanical operations and notifications to the driver. Other inputs come from sensors within the car that monitor the state of the driver, in case there is a need for an emergency override of the rest of the system.



1. Select and use an appropriate UML behavioural diagram, between use-case, state, sequence and activity diagrams, to describe the behavior of the action engine module **[5]**

   Ans.

   -selection of suitable modeling diagram and rationale (1): any of the three diagrams are suitable, given the little amount of information given about the functionality of the action engine. So, any well motivated diagram type is fine.

   -model sketch (4): marks awarded in terms of clarity, use of given information about the action engine module (e.g action tasks as shown on the functional diagram include warn, complement, and control(e.g braking etc)) and creativity (ability to think of other relevant detail not included in the question)

2. Compare and contrast the computational characteristics of the *sense*, *understand* and *act* functional modules of the system in terms of computational complexity, throughput and latency requirements **[6]**

	sense	**understand**	**act**
computational complexity:	-relatively simple algorithms -largely data-indepedent operations -e.g filtering	-relatively complex algorithms -high data-dependent operations -e.g deep learning feature extraction algorithms	-medium to low -medium data-dependent operations -e.g emergency braking action
throughput:	-very high throughputs -e.g very high speed and high resolution cameras and GPS for real-time sensing of the environment	-high throughputs -very high sensory input throughputs reduced as data travels down the front-end pre-processing stages	-moderate throughputs
latency:	-short (for RT monitoring)	-medium (for RT monitoring)	-relatively longer (for RT control)

3. Multicore GPPs, DSPs, GPUs and FPGAs are the main programmable hardware solutions used to implement applications with high processing requirements. Propose and describe a specific processing hardware architecture you would use to implement the computational functions of the autonomous vehicle system shown in the figure above. You should explain the level of parallelism possible for, and what programmable hardware will be used to implement the sensor processing, sensor fusion and action engine functions of the system. Include brief justifications for your architectural decisions. Use explanations and a well labelled block diagram (or UML deployment diagram) to describe your proposed architecture. **[12]**

   Ans.

   -brief characterisation of the various system functional blocks (FE sensor processing, BE sensor fusion, BE action engine) in terms of possible parallelism ***(3 marks)***

   -a sketchy block diagram showing main functional blocks and the proposed types of processing platforms ***(3 marks)***

   -diagram should be accompanied by a brief rationale for the choices of appropriate processing platforms for each functional stage ***(6 marks)***

   -block diagram should also show also example interconnection technology to be used to interface the various processing modules (i.e for a heterogeneous processing architecture)

	sense	**understand**	**act**
parallelism:	-mainly data-parallel -task-parallelism across multiple sensor pre-processing stages -e.g kernels include filtering for sensor input conditioning	-mix of data and task-parallelism -e.g on-going AI model training operations	-mix of data and task parallelism -but mainly task-parallelism -e.g sending out multiple control signals to multiple MCUs and they in turn performing different actions concurrently to implement a particular main action
processing hardware:	-FPGAs or ASICs -(FPGAs) best for high throughput parallel data streams processing -both capable of very high processing throughputs	-any combination of the microprocessor platforms -DSPs, GPPs, GPUs -They offer good support for control-oriented (multicore GPPs), high precision calculations (DSPs) operations and also good support for data-parallelism (GPUs)	-mixture of FPGAs for glue logic -MCUs for interfacing with and control of mechanical components

QUESTION THREE [19 MARKS]

In mathematics, the Sieve of Eratosthenes is a simple, ancient algorithm for finding all prime numbers up to any given limit. Consider the pseudocode of the algorithm given below.

```
Input: an integer n > 1.
Let A be an array of Boolean values, indexed by
integers 2 to n, initially all set to true.
for i = 2, 3, 4, ..., not exceeding n:
  if A[i] is true:
    for j = i^2, i^2+i, i^2+2i, i^2+3i, ..., not exceeding n:
      A[j] := false.
Output: all i such that A[i] is true.
```

1. Write an Octave function that implements the functionality of the Sieve of Eratosthenes **[7]**

2. Explain how you would develop a multi-threaded implementation of the Sieve of Eratosthenese algorithm. You need to explain and motivate what partitioning strategy you would use so as to divide the work efficiently across the threads, how you would get the input integer n to the threads, and how you would implement the necessary thread kernel(s) (**NB**: you can just write C code or you can explain in English and use diagrams to illustrate) **[12]**

   Ans:

-passing info to threads (2)

-partitioning strategy (3)

-synchronisation (2)

-kernel implementation (5)

```
struct thread_data{
  int thread_id, start_index, stop_index;
};
...
void *SieveOfEratosthenes_Thread(void *threadarg) {
  struct thread_data *my_data;
  my_data=(struct thread_data*) threadarg;
  int tid=my_data->thread_id,
  t_starti=my_data->start_index,
  t_stopi=my_data->stop_index;
  for (int p=t_starti; p<=t_stopi; p++) {
    // If prime[p] is not changed, then it is a prime
      if (prime[p]==true) {
      // Update all multiples of p
        for (int j=p*p; j<=n; j += p)
          prime[j] = false;
        }
  }
}
```

QUESTION FOUR [9 MARKS]

1. The Core i5 Ivy Bridge, released in 2012, had a clock rate of 3.4 GHz and voltage of 0.9 V. Assume that, on average, it consumed 30 W of static power and 40 W of dynamic power. Find the average capacitive load for the Core i5 processor. **[4]**

   Ans:

   - $DynamicPower = \frac{1}{2} . C . V^2 . F$

   - $40 = \frac{1}{2} x C x 0.9^2 x 3.4 x 10^9$

   - $C = 0.029 pF$

2. Implementations of floating-point (FP) square root vary significantly in performance. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical benchmark on a multiprocessor. One proposal is to add a custom FPSQR chip that will speed up this operation by a factor of 8. The other alternative is just to try to make all FP instructions run faster by improving the underlying architecture of the multiprocessor so that it has a lower CPI; FP instructions are responsible for a total of 50% of the total execution time. The design team believes that they can halve the CPI of the machine and also make all FP instructions run three times faster with the same effort as required for the fast square root.

Compare the two design alternatives **[5]**

Ans:

-We compare the two design alternative in terms of their achievable speedup factors

-$Speedup_{with-FPSQR-HW} = \frac{1}{(1-p)+\frac{p}{s}} = \frac{1}{(1-20\%)+\frac{20\%}{8}} = 1.21$

-$Speedup_{with-Optimised-FP-Ops} = \frac{1}{(1-p)+\frac{p}{s}} = \frac{1}{(1-50\%)+\frac{50\%}{3}} = 1.5$

-Therefore, optimisation of the FP instructions will yield better performance than inclusion of a custom FPSQR chip in terms of response time

CHEAT SHEET

Parallel performance

Amdahl's law:

$$S = \frac{1}{(1-p) + \frac{p}{s}} \tag{1}$$

BCE-based Speedup on symmetric multicore:

$$Speedup_{symmetric} = \frac{1}{\frac{1-f}{perf(r)} + \frac{f.r}{perf(r).n}} \tag{2}$$

BCE-based Speedup on dynamic multicore:

$$Speedup_{dynamic} = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{n}} \tag{3}$$

The classic CPU performance equation:

$$CPU_{clockcycles} = \sum_{i=1}^{n} CPI_i \,.\, IC_i \tag{4}$$

$$CPU_{time} = (\sum_{i=1}^{n} CPI_i \,.\, IC_i) \,.\, ClockCycle_{time} \tag{5}$$

$$CPI = \frac{\sum_{i=1}^{n} CPI_i \,.\, IC_i}{Instructioncount} \tag{6}$$

Processor Dynamic power:

$$DynamicPower = \frac{1}{2} \,.\, C \,.\, V^2 \,.\, F \tag{7}$$

Pthreads

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);
