High Performance Embedded Systems EEE4120F

2021 FINAL EXAM

9 July 2021

ANSWERS MEMO

Section 1

Question 1.1

(a) Expected to discuss the affordances that a heterogeneous architecture provides: some processors that are suited to course-grained parallel solutions (e.g. SMP and GPUs, e.g. SIMD allowing the same operation to run on multiple data) and others suited to fine-grained parallel solutions, such as FPGAs doing many different computing tasks at the same time.

[2]

(b) Difficulties that the programmers are likely to need expertise in using a variety of programming tools, such as CUDA, pthreads, MPI, etc., which could make it more difficult and expensive obtaining staff to develop applications, and also could cause lengthier development times, such as developers needing to learn tools in order to leverage the available processing power of the cluster. Maintaining the cluster may be more challenging and expensive as well, for example being able to service the variety of architectures from SMP to FPGAs. Additionally there can also major difference between the application design approaches, and programming paradigms, for example the way of thinking about SMP-based application design and use of pthreads is quite different to the design thinking for Cell processors and FPGAs. Furthermore considerations for specialized I/O system and performing I/O on the different architectures could be quite different and more difficulty to accomplish than when using only standard programming languages such as C and Python.

[4]

(c) The program is not using adequate semaphores to protect the value of sum that is being written by multiple threads which are contending on the write access to sum. In some cases multiple threads may have read a same value from sum, ending up with stale values being overwritten and thus calculating the wrong final value. It is also slower due to the overhead of starting the threads and context switching.

[4]

Question 1.2

Q1.2(a) Can you implement this recursive function in parallel? If yes explain how and whether it's a good idea. If no explain why not.[4]

ANS: Yes, one thread per recursive step.
How: Where the thread waits for the returned value from the next thread.
It is a bad idea as each thread will be created only to wait for the output of the next thread that is created, ultimately using up resources unnecessarily. Therefore, it is vastly more efficient to run this function sequentially.

Q1.2(b) Is this algorithm highly *parallelizable*? [1]

ANS: Yes

Q1.2(c)
If you were to implement this algorithm using openCL how would to setup the memory management, with you work items and work groups? No, code is required but emphasis should be placed on efficiency and you can use the example above to aid in your explanation. [5]

ANS:

- Each factorial function will have its own workgroup, making 3 work groups in the above example.
- Each positionpower multiplication can have it own kernel, making up the work-item.
- The positionpower work items output will be stored in the local memory of the work group.
- One work item in each work group will then multiply all the positionpower work items outputs for that group together and pass the value to global memory.
- The CPU will then add the values in the global memory together.
- *Note:* there are numerous correct answers

Question 1.3

Q1.3(a)
i. The main difference between a FPGA and a PLA is the: the architecture (how the system is configured and programmed), the number of logic elements available, and the programming speed. The FPGA has a more complex architecture, supports more complex designs, usually many types of logic elements. [2]
ii. There is usually a particular programing sequence needed for an FPGA. In particular, if a FPGA board needs to start up without being programmed from a host (e.g. attached PC), there needs to be some way to program the FPGA. This is where a configuration architecture, utilizing a statemachine implemented using a PLA or CPLD, is used in order to read the FPGA program from non-volatile memory (e.g. a EEPROM chip) and to program the FPGA. Furthermore, the PLD/CPLD may also include logic to support programming from a host, i.e. to receive a program sent from the host into a then exercise the necessary programming pins on the FPGA in order to program it. [3]
Q1.2 (b)
Difficulties associated with taking an FPGA design forward to an ASCI design include accounting for differences in propagation delays and operational speeds, different layouts of components; possibly different implementations of components or CLBs that are utilized. Changes in the interconnections and electrical properties of the material used for the ASIC. Futhermore, the tool chains may be quite different and require the designer to undergo a lengthy learning curve to learn how to use the tools effectively. There would also need to be more reliance on simulation, due to the

expense of running of physical instances of ASICs; whereas for FPGAs it is just a matter of programming the FPGA and testing it on hardware, using a development kit prototyped board. Risks for ASIC include the potential for having a re-do designs and the expense of additional runs to compensate for design faults. Further there may be the risk of hiring consultants to assist with ASIC design and that it is difficult to predict how long it will take to achieve a final operational ASIC due to the complexity of this practice. [4]
Q1.2 (c) Advantages of parallel code are: potential for increased performance (by doing multiple operations in parallel as opposed to being limited to sequential operation), the potential for redundancy and fault tolerance (e.g. running the same operation on multiple different processors which could be used to work around interference or damage that could cause processors to fail temporarily or permanently). Improved responsiveness / decreased latency, the ability to respond to interrupts more quickly, without necessarily relying on one available processor to handle the request. [3]

Section 2

Q2.1 (e)
Q2.2 (d) Clearly R&D costs is the most significant Non-recurring cost, for a particular model at least.
Q2.3 (c)
Q2.4 (c)
Q2.5 (e)
Q2.6 (c) The interlaced memory partitioning approach is used.
Q2.7 There are true/false options for these 5 questions, the answers are:

(i) T: yes, that's the concept of a golden measure as explain at the start of the course
(ii) F: no, actually a perfect correlation between two identical data sets returns the value 1
(iii) T: yes, a DWARF relates to a type of common processing pattern found in HPEC system
(iv) F: no – SWAP = Size Weight And POWER
(v) T: yes - DRYSTONES related to iterations of a performance algorithm rather than a measure of operations per second.

Section 3

Q 3.1 [20 marks]

(a) For this section the students may decide to draw out the stations and determine where the bottlenecks are, particularly the pourer that is limited to three top molds giving 1.5 top molds per minute (the slowest one). The assembly line is 12 workstations, so that is 12 top molds and 12 bottom molds per 15 minutes, consuming 12 top molds per 15 minutes. So the max throughput is 0.8 calculators per minute, the number of workspaces being the main bottleneck. [8]
(b) Should determine the 0.8 calculators per minute, or 48 calculators per hour. Indicate required rates of production at the different stations. [7]
(c) Student to provide an argument as to which stages could be expanded [5]

Q 3.2 [15 marks]

A solution such as the below would suffice:

```
module hugecalc (lev,acc,dsp,add,sub,mul,eq);
 /* FPGA prototyped version of the Huge Calculator */
 input  lev[31:0]; /* last entered value */
 inout  acc[31:0]; /* accumulator */
 output dsp[31:0]; /* display register */
 input  add,sub,mul,eq; /* the math function buttons */
 /* add your code here: */
 always@ (posedge add or posedge mul or posedge eq)
 begin
   if (add == 1) begin
      acc = acc + lev;
      dsp = acc;
   end
   else if (sub == 1) begin
      acc = acc - lev;
      dsp = acc;
   end
   else if (mul == 1) begin
      acc = acc * lev;
      dsp = acc;
   end
 end // always@
endmodule
```