EEE4084F Quiz 1 – 2013

SAMPLE SOLUTIONS

Q1.

The main difference between the two is that a reconfigurable computer can dynamically change aspects of its hardware as well as offering the flexibility of software running on CPUs, whereas a software processor based system has a static processing hardware architecture also with the advantage of software. Reconfigurable computing is effectively a hybrid approach of computing that provides advantages of making better use of the speed of hardware solutions, and also being able to dynamically configure hardware parts of the systems (in particular interconnects between processing elements) to adapt to application needs. A major advantage of the software processor based architectures is generally the advantages of lower cost and easier software programmability. The table below summarizes the advantages of each approach:

Reconfigurable computing advantages	**Software processor system advantages**
Faster than software alone	Flexibility of software
More flexible than software	Adaptable (able to change programs quickly)
More flexible than hardware	Can be much cheaper (often easier design work)
Allows for highly parallelized / fine grained processor node design	Reuse of a huge amount of code that's out there.

Note to marker: of course I don't expect such a detailed and wordy answer from students; but this sample solution touches on the main points so of which students should have mentioned.

Q2.

(a) SMP = Symmetric Multi-Processor. The main characteristic of a SMP processor design is a) there are multiple processor cores / CPUs of the same design (i.e., symmetric) within the processor chip, and these processors and their (usually shared) memory is all on a common front side bus (FSB), i.e. a bus that connects the CPU and motherboard subsystems. The processors are all synchronously clocked, and each core can be executing a different instruction at the same time.

(b) Automatic parallelization refers to a program that automatically translates sequential code (i.e., a program that provides a sequential solution to a problem) into parallelized code (i.e., a program that provides a solution that incorporates concurrency). Generally, a compiler or interpreter is not likely to be the way to create these parallel solutions, but rather a higher level application with more 'intelligence' would be needed that somehow understands the problem and design pattern needs, and then generates a code solution that is passed to compilers, linkers and the like.

(c) Some points:

- Difficulty in figuring out data dependencies
- Deciding when to implement semaphores and locks
- How to split-up a loop into parallel parts
- Deciding how to break-up data to distribute

- Identifying and dealing with data hazards
- When code needs to block and when not
- Figuring out timing dependencies
- Having to convert clocks of statements or functions in inter process calls
- Basically, being able to take on some of the 'intelligence' that a human programmer uses in order to change a sequential program into a parallel one.

Q3.

a) B. Refers to time in the real world
b) B. Interleaved, because small consecutive segments at a time as opposed to rows at a time.
c) C

Q4.

a) <u>Temporal paradigm</u> = describing computation in order of time, one operation after another. Adv: Easier to write, to explain or make sense of, and for sharing solutions. Disadvantage: not so effective for describing parallel operations, more suited to sequential coding. <u>Spatial paradigm</u> = computation expressed in spaces and relations between operations. Effective / more intuitive for expressing parallel solutions. Adv: Allows effective methods of abstraction, particularly abstracting out time ordering and focusing rather on data dependencies. Disadv: can be difficult to handle the details of large solutions, can be more difficult to make sense of, or to share solutions, than the case for the temporal paradigm.
b) Golden measure = A (usually sequential) solution to the problem developed as a 'yard stick' to validate and contrast the operation of alternate (possibly better performing) solutions. It could be a solution that runs slowly, isn't optimized, but it gives an accurate result (a computation result to aim form, as opposed to reductions in the size/speed that is wanted).

   To spot mistakes in a parallel solution, the outputs of the program under test is compared to that of the golden measure. This could be done by inspecting values by eye, using graphs to visually contrast the results or statistical operations such as correlation to judge that the accuracy of computed solutions remain within tolerated bounds.

Note to marker: of course such a long wordy answer isn't expected; it is more the keywords and the gist of the explanation that matters.

c) Reasons for most code not being parallel is that: 1) The effort in reworking the code might be too much for the negligible or hardly noticeable performance improvement of the computation. 2) There might be no sufficiently pressing need to make the program run faster or operate with concurrently executing subtasks (i.e., inefficient utilization of the processing nodes might be of no concern). 3) The algorithm(s) concerned might have no parallel solutions (i.e., data dependencies restricting parallelization). … that's just some of many other reasons.

Q5.

a) I'll list most of them:

- What are the applications?
- What are common kernels of the applications?
- What are the hardware building blocks?
- How to connect them?
- How to describe allocations and kernels?
- How to program the hardware?
- How to measure success?

b) A + B + C (i.e. all but D)

c) Conventional Wisdom (CW) = concept or understanding that is generally accepted as valid by experts in the field. These explanations are general considered as valid beyond further explanation or debate by the *hoi polloi* (to be more precise: at least the general populace who are willing to accept what the experts have decreed).

I've dumped the whole lot of the old CW vs. new CW from the paper (in blue so you can skip the possible answers for this question quickly):

1. Old CW: Power is free, but transistors are expensive.
   • New CW is the "Power wall": Power is expensive, but transistors are "free". That is, we can put more transistors on a chip than we have the power to turn on.
2. Old CW: If you worry about power, the only concern is dynamic power.
   • New CW: For desktops and servers, static power due to leakage can be 40% of total power.
3. Old CW: Monolithic uniprocessors in silicon reliable internally, errors occurring only at the pins.
   • New CW: As chips drop < 65nm feature sizes, they will have high soft + hard error rates.
4. Old CW: By building upon prior successes, we can continue to raise the level of abstraction and hence the size of hardware designs.
   • New CW: Wire delay, noise, … and so on conspire to stretch the development time and cost of large designs at 65 nm or smaller feature sizes.
5. Old CW: Researchers demonstrate new architecture ideas by building chips.
   • New CW: The cost of masks at 65 nm feature size, the cost of Electronic Computer Aided Design software to design such chips, and the cost of design for GHz clock rates means researchers can no longer build believable prototypes…
6. Old CW: Performance improvements yield both lower latency and higher bandwidth.
   • New CW: Across many technologies, bandwidth improves by at least the square of the improvement in latency. [Patterson 2004]
7. Old CW: Multiply is slow, but load and store is fast.
   • New CW is the "Memory wall" [Wulf and McKee 1995]: Load and store is slow, but multiply is fast. Modern microprocessors can take 200 clocks to access DRAM, but …
8. Old CW: We can reveal more instruction-level parallelism (ILP) via compilers and architecture innovation. Examples from the past include branch prediction…

- New CW is the “ILP wall”: There are diminishing returns on finding more ILP.

9. Old CW: Uniprocessor performance doubles every 18 months.
    - New CW is Power Wall + Memory Wall + ILP Wall = Brick Wall.
10. Old CW: Don’t bother parallelizing your application, as you can just wait a little while and run it on a much faster sequential computer.
    - New CW: It will be a very long wait for a faster sequential computer (see above).
11. Old CW: Increasing clock frequency is the primary method of improving processor performance.
    - New CW: Increasing parallelism primary method of improving processor performance.
12. Old CW: Less than linear scaling for a multiprocessor application is failure.
    - New CW: Given the switch to parallel computing, any speedup via parallelism is a success.

Q6.

a) – ignore this one; was a duplicate question.

b) A major advantage for this that the spatial paradigm provides is the expression of data dependence, together with emphasising areas of communication between the operations that occur in the space. It also gives the potential advantage to identifying hardware or computation resource needs, that can be used to expresse a more hardware-centred and thereby more parallel solution to a computation problem.

Q7.

```
function [ret] = sort (x)
  # x : an array of values that can be ordered
  # ret : array to be returned
  swapped = true;
  N = length(x);
  while (swapped)
    swapped = false;
      for i=2:N
         if (x(i-1) > x(i)) # see if pair is out of order
            tmp = x(i-1);
               x(i-1)=x(i);
               x(i)=tmp;
         endif
      endfor
  endwhile
  ret = x;
endfunction

(note – I’ve tested the above in Octave and it works fine. But do change the name
from sort to sort1 to avoid confusion with the build in sort command so that the
function can be tested).
```

Q8. C.