Transactions on a (Familiar) Doubly Linked List

Problem 1. (30 points):

Consider a SORTED doubly-linked list that supports the following operations. Yes, this is the same data structure as used in Written Assignment 4.

- **insert_front**, which traverses the list from the front.
- **delete_front**, which deletes a node by traversing from the front
- **insert_back**, which traverses the list **backwards from the end** to insert a node in the opposite order as insert_front.

In this problem, assume that the entire body of each function insert_front, delete_front, and insert_back is placed in its own atomic block, and the code is run on a system supporting optimistic (for reads and writes) transactional memory.

A. (6 pts) Your friend writes three unit tests that each execute a pair of operations concurrently on the list shown above.

- Test 1: insert_front(2), delete_front(14)
- Test 2: insert_front(12), delete_front(6)
- Test 3: insert_front(13), insert_back(4)

Assuming all unit tests start with the list in the state shown above, is the code correct? (By correct, we mean there are no race conditions and so all operations will modify the data structure according to their specification.) Why or why not?
B. (8 pts) Consider two transactions performing \texttt{insert\_front(4)} and \texttt{delete\_front(14)}. Assume both transactions start at the same time on different cores and the transaction for \texttt{insert\_front(4)} proceeds to commit while the \texttt{delete\_front(14)} transaction has just iterated to the node with value 7. Must either of the two transactions abort in this situation? Why? (Remember this is an optimistic transactional memory system!)

C. (8 pts) Must either transaction abort if the transaction for \texttt{delete\_front(14)} proceeds to commit before the transaction for \texttt{insert\_front(4)} does? Why? Please assume that at the time of the attempted commit, \texttt{insert\_front(4)} has iterated to node 3, but has not begun to modify the list.
D. (8 pts) Must either transaction abort if the situation in part C is changed so that `delete_front(14)` attempts to commit first, but by this time `insert_front(4)` has made updates to the list (although not yet initiated its commit)? Why?
Paparazzi Camera

Problem 2. (40 points):

You are designing a heterogeneous multi-core processor to perform real-time “celebrity detection” on a future camera. The camera will continuously process low-resolution live video and snap a high-resolution picture whenever it identifies a subject in a database of 150 celebrities. Pseudocode for its behavior is below:

```c
void process_video_frame(Image input_frame) {
    Image face_image = detect_face(input_frame);
    for (int i=0; i<150; i++)
        if (match_face(face_image, database_face[i]))
            take_high_res_photo();
}
```

In order to not miss the shot, the camera MUST call `take_high_res_photo` within 500 ms of the start of the original call to `process_video_frame`! To keep things simple:

- Assume the code loops through all 150 database images regardless of whether a match is found (e.g., we want to find all matches).
- The system has plenty of bandwidth for any number of cores.

Two types of cores are available to use in your chip. One is a fixed-function unit that accelerates `detect_face`, the other is a general-purpose processor. The cost (in units of chip resources) of the cores and their performance (in ms) executing important functions in the pseudocode are given below:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Core Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1 (fixed-function)</td>
</tr>
<tr>
<td>Resource Cost</td>
<td>1</td>
</tr>
<tr>
<td>Perf (ms): detect_face</td>
<td>50</td>
</tr>
<tr>
<td>Perf (ms): match_face</td>
<td>N/A</td>
</tr>
</tbody>
</table>

A. (10 pts) Assume a video frame arrives exactly every 500 ms. **If you only use cores of type C2**, what is the minimum number of cores do you need to meet the performance requirement for the video stream? (You cannot change the algorithm, and please justify your answer).
B. (10 pts) Your team has just built a multi-core processor that contains a large number of cores of type C2. It achieves $15.2 \times$ speedup on the camera workload discussed above. Amdahl’s Law says that the maximum speedup of the camera pipeline in this problem should be $(250 + 150 \times 25)/250 = 16 \times$, so your team is happy. They are shocked when your boss demands a speedup of $20 \times$. Your team is on the verge of quitting due “unreasonable demands”. How do you argue to them that the goal is reasonable one if they consider all the possibilities in the above table? (Hint: What assumption are they making in their Amdahl’s Law calculations, and why does it not hold?)

C. (10 pts) Now assume you can use both cores of types C1 and C2 in your design. How many of each core do you choose to minimize resource usage, while still meeting the same latency requirements as in part A? Does your new chip use more or fewer total resources than your solution in part A (by how much)?
D. (10 pts) Beyonce is about to drop a new album, and your paparazzi customers want to follow her around all day. They request a camera that is more energy efficient. Energy efficiency is so important they are willing to relax their performance requirement and allow high-res photos to be taken within 2500 ms (not 500 ms) of a video frame arriving. You find that nearly all the power in your application is used by loading database faces from memory. Describe how you would change the pseudocode to **approximately triple** the energy efficiency of the camera while still meeting the performance requirements. You should assume you use the same processor design as in part C (or part A for that matter), and that your processor has a cache that holds up to four database images. Hint: we are expecting you to reduce the number of times `database_face[i]` is loaded from memory by accessing data resident in cache.
Tricky Little Graphs

Problem 3. (30 points):

```c
struct Graph_node {
    Lock  lock;
    float value;
    int   num_edges; // number of edges connecting to node (its degree)
    int*  neighbor_ids; // array of indices of adjacent nodes
};

// a graph is a list of nodes, just like in assignment 3
Graph_node graph[MAX_NODES];
```

Consider the undirected graph representation shown in the code above.

Your boss asks you to write a program that atomically updates each graph node's `value` field by setting it to the average of all the values of neighboring nodes. The program must obtain a lock on the current node and all adjacent nodes to perform the update. It does so as follows...

```c
void update(int id) {
    Graph_node* n = &graph[id];
    LOCK(n->lock);
    for (int i=0; i<n->num_edges; i++)
        LOCK(graph[n->neighbor_ids[i]].lock);
    // now perform computation...
}
```

Consider running the update code in parallel on nodes 0 and 1 in the two graphs below. For each graph, determine if deadlock occurs. Please describe why or why not. (Note: we do not ask you to solve the deadlock problem, but think about you might avoid it, assuming you must still only use locks. Consider changing the order in which you take the locks.)