System-on-Chip (SoC) Design

EE382M.20, Fall 2018

Homework #2

Assigned: September 25, 2018
Due: October 11, 2018

Instructions:

- Please submit your solutions via Canvas. Submissions should include a single PDF with the writeup and single Zip or Tar archive for source code.
- You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions.

Problem 1: WCET Estimation (35 points)

Consider the `gemmA` code, matrix data and cache architecture from Homework 1, Problem 1c).

(a) How often is each line of code/statement executed for the given *M*, *N* and *K*?

(b) What is the Must-Cache state when executing the inner-most loop body computing C[i][j]? What is thus the worst-case cache hit rate to be assumed for safe and conservative WCET analysis? How does that compare to the actual cache hit rate you computed in Homework 1?

(c) To improve tightness of the cache and WCET analysis, one can partially unroll loops. Assuming just 1 unrolling for the inner-most loop as follows:

```
gemmA: for (int i=0; i< M ; i++)
         for (int j=0; j< N; j++) {
           C[ i ][ j ] += A[ i ][ 0 ] * B[ 0 ][ j ];
           for (int k=1; k < K; k++)
             s: C[ i ][ j ] += A[ i ][ k ] * B[ k ][ j ];
           }
```

What is now the Must-Cache state when executing the inner-most statement *s*? What is thus the estimated cache-hit rate under worst-case assumptions?

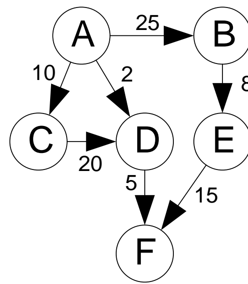
(d) Assuming a straightforward ARM assembly implementation of the inner-most GEMM loop as shown below, what is the WCET estimate for this code snippet when running on a micro-architecture with a typical single-issue 5-stage in-order pipeline (consisting of fetch, decode, execute, memory and write-back stages), full data forwarding/bypassing, static not-taken branch prediction with branches resolved in the execution stage, and a blocking cache with zero hit latency and a cache miss penalty of 10 cycles?

```
; Assume base pointers for C, A and B are in R0, R1 and R2
; R10, R11 and R12 hold index variables i, j and k
     MOV R12, #0          ; R8 holds k
     MUL R3,R11,#6        ; compute address of C[i][j]
     ADD R3,R3,R10
     LDR R4,[R0, R3 LSL #2] ; and load C[i][j]
l3:  ADD R12,#1
     CMP R12,#4
     BEQ end
```

```
         MUL R5,R12,#6        ; compute address of A[i][k]
         ADD R5,R5,R10
         LDR R5,[R1, R5 LSL #2] ; and load A[i][k]
         MUL R6,R11,#6        ; compute address of B[k][j]
         ADD R6,R6,R12
         LDR R6,[R2, R6 LSL #2] ; and load B[k][j]
         MUL R5,R5,R6    ; multiply
         ADD R4,R4,R5    ; accumulate
         B l3
    end: STR R4,[R0, R3 LSL #2] ; write back C[i][j]
```

Problem 2: Partitioning (30 points)

Consider the following task graph where communication costs indicate the number of kBytes transferred between tasks:



(a) Apply a hierarchical clustering algorithm where the communication cost of a clustered node is the sum of the bytes exchanged with other nodes. Show the graphs with communication costs after each clustering step. What is the final HW/SW partition on a system with one CPU and one hardware accelerator? What is the partition on a three-processor system (one CPU, one DSP and one accelerator)?

(b) Apply the (full) Kernighan-Lin algorithm to partition the graph into two groups with an equal number of nodes, starting from an initial A,B,C and D,E,F binning. Note that, as discussed in class, the full Kernighan-Lin algorithm looks at complete sequences of node swaps. In each iteration of the algorithm, a whole set of possible partition candidates is constructed by consecutively swapping nodes that have not been swapped before and that result either in the largest gain or least loss in inter-partition communication cost per swap (i.e. considering intermediate swaps that may increase cost). The set of candidates is complete when all nodes have been considered for swapping, i.e. the graph is mirrored. Out of this set of candidate partitions, the algorithm selects the partition with the least cost, i.e. it actually only executes the partial subsequence of swaps that leads to the largest overall reduction in cost. This process (of constructing candidates and selecting the best) is repeated until no more gains can be achieved (there is no candidate that leads to any reduced cost).

(c) Which algorithm gives the better solution for a 2-processor system? What are the tradeoffs between the two solutions? Is there a better 2-processor solution that minimizes communication costs?

Problem 3: Scheduling (35 points)

Consider the problem of scheduling the following sets of aperiodic, independent tasks:

Task	Arrival Time	Deadline	Execution Time
A	0	3	1
B	0	7	2
C	2	6	2
D	3	5	1

(a) Apply an EDF algorithm and show the resulting schedule for executing this set of tasks.

(b) What is the maximum execution time of task C for this task set to remain schedulable?

(c) Now assume a system with 2 (homogeneous) processors/cores. What is the maximum execution time of task C for this set to be schedulable under a global EDF strategy in which tasks can migrate between processors/cores freely, i.e. strictly following a strategy in which at any point in time the two tasks with the highest priority are running? What is the maximum time of C under a partitioned EDF strategy in which tasks are assigned to fixed processors/cores in a pre-defined manner? Show the schedule in each case.