System-on-Chip (SoC) Design

EE382M.20, Fall 2021

Homework #2

Assigned: September 23, 2021
Due: October 7, 2021

Instructions:

- Please submit your solutions via Gradescope. Submissions should include a single PDF with the writeup.
- You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions.

Problem 1: Execution Time Estimation (30 points)

Consider the following compiler-generated intermediate representation code for the inner-most loop of a 3x3 GEMM:

```
oa = i*3;
ob = j;
oc = i*3 + j;
c = C[oc];
for (int k=0; k < 3; k++, oa += 1, ob += 3)
     c += A[oa] * B[ob];
C[oc] = c;
```

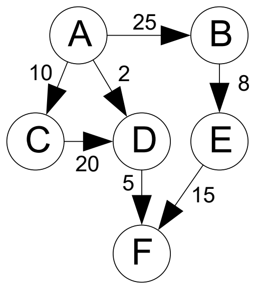
(a) Given an operator library that supports 2-input, 1-output operations `LD` (Indexed Load *a*[*b*] from memory given base address *a* and word offset *b*<<*2*), **+** (Addition, *a*+*b*), ***** (Multiplication, *a***b*), **<** (Compare returning a true result if *a*<*b*, false otherwise) as well as a 1-input **🡐** (register initialization/assignment with constant) and a 3-input `STR` (Indexed Store to memory from base address *a* and offset *b*<<*2*), show the control-dataflow graph (CDFG) for the code above. Assume that `A`, `B` and `C` matrices are in memory with given base addresses, and variables `i` and `j` are already initialized in registers.

(b) Assuming a processor with a SIMD datapath (e.g. ARM NEON) that can perform (issue) up to four independent `LD`, 🡐, +, *, < or `STR` operations per cycle (`LD`/`STR` must have same base address), how many cycles are required to finish the inner-most GEMM loop? Assume assignments, loads and stores take 1 cycle (all data is in the cache), add, compare and multiply take 1, 1 and 3 cycles, respectively, and a 100% accurate branch predictor.

(c) Now apply loop unrolling optimizations to the inner-most GEMM code. Show the CDFG when fully unrolling the loop. How many cycles does it require to execute the unrolled code?

Problem 2: Partitioning (40 points)

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


$$\begin{tikzcd}
 & A \arrow[r, "25"] \arrow[dl, "10"'] \arrow[d, "2"] & B \arrow[d, "8"] \\
C \arrow[r, "20"'] & D \arrow[d, "5"'] & E \arrow[dl, "15"] \\
 & F &
\end{tikzcd}$$

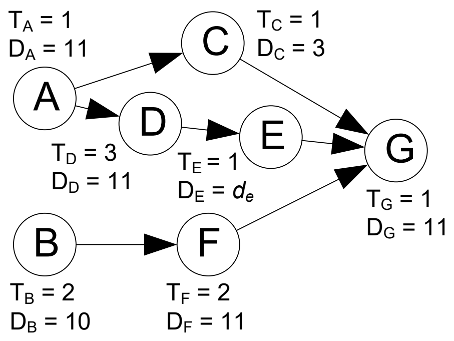
(a) Apply a hierarchical clustering algorithm where the communication cost of a clustered node is the sum of the bytes exchanged with other nodes. Show the graphs with communication costs after each clustering step. What is the final HW/SW partition on a system with one CPU and one hardware accelerator? 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 (30 points)

Consider a system that periodically executes the following graph of tasks with dependencies, (precedence constraints). Due to the dependencies, all tasks need to run at the same rate with a common period of 11 while all precedence relationships are maintained within each period. In addition, tasks individually have stricter deadlines (relative to the start of the graph's period). Task execution times T_i and relative deadlines D_i are as indicated in the graph. Assume that other than dependencies, all tasks are ready to execute at the beginning of each period, i.e. have arrival times of zero (relative to the start of the graph's period).



(a) What is the smallest deadline d_e of task E for which the graph is schedulable on a single processor using a modified EDF* algorithm that accounts for dependencies? Show the resulting schedule for one period, i.e. for one execution of the graph.

(b) What is the smallest deadline d_e of task E for which the graph is schedulable when executed on two processor strictly following a global EDF* algorithm in which tasks can freely migrate between processors and at any point the two ready tasks with the highest priority are running? Show the resulting schedule for one period.

(c) Can a smaller d_e be achieved on one or two processors using a different schedule? If so, show the schedule and achievable d_e. If not, will EDF* always be able to find the tightest uni- or multi-processor schedule that exists? Explain why or show a counter-example (e.g. a modification of the graph above for which EDF* does not achieve the smallest d_e).