System-on-a-Chip (SoC) Design

EE382V, Fall 2014

Homework #2

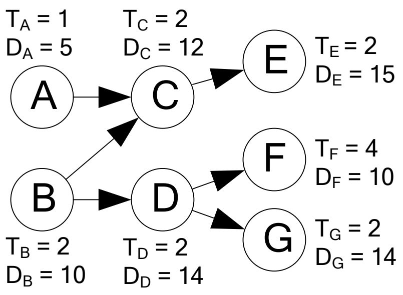
Assigned: September 25, 2014
Due: October 12, 2014

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: Task Scheduling (50 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 15 while all precedence relationships are maintained within each period. In addition, however, tasks may 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 tasks *A* and *B* are ready to execute at the beginning of each period.



(a) What is the utilization of a single processor running the tasks?

(b) Apply an EDF algorithm and show the schedule for one period (relative to the start of the period), i.e. for one execution of the graph.

(c) In the presence of dependencies, EDF is no longer optimal in guaranteeing to find a schedule if it exists. However, a modified EDF* strategy becomes optimal by adjusting the deadlines of individual tasks to take their successors into account. This is done by starting with the sinks of the graph (nodes with no successors) and successively propagating deadlines that are adjusted for execution times upwards through the graph. Every time a deadline is propagated to a predecessor, the execution time of the current node is subtracted (such that it is guaranteed that the node will have enough time to execute once the predecessor has finished). At each node, a new deadline is then computed to be the smaller of its original deadline and of the minimum over adjusted deadlines propagated upwards from all its successors. Indicate the dependency-adjusted deadlines in the original task graph above and show the resulting EDF* schedule.

(d) Show the EDF* schedule for the task graph with adjusted deadlines executed on two processors. Assume that tasks can migrate between processors freely, i.e. strictly follow a strategy in which at any point in time the two tasks with the highest priority are running.

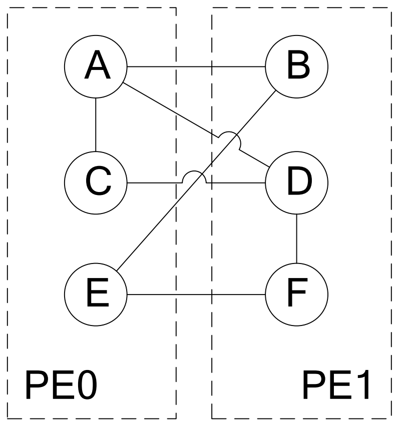
(e) Does any uni-processor, priority-based scheduling of tasks with dependencies ever require preemption? If so, under what conditions? If not, why not? How about in priority-based multi-processor scheduling?

(f) EDF-type scheduling algorithms are only optimal in uni-processor systems. In systems with multiple processors/cores, there is no known optimal, tractable (polynomial complexity) scheduling strategy. What can be done, however, is to statically partition tasks onto processors in a fixed, pre-defined manner and then run a regular scheduling algorithm on each processor. Assuming no precedence constraints and no deadlines other than the end of the period, how many processors are needed to run the given task set in a period of 5 time units? Show a possible assignment of tasks to processors.

Problem 2: Task Partitioning (50 points)

In class and the homework we only looked at a simplified version of the Kernighan-Lin algorithm. The full Kernighan-Lin algorithm looks at complete sequences of node swaps. In each iteration of the algorithm, a 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).

Consider the following task graph with uniform communication costs per edge and initial partitioning:



(a) Apply the full Kernighan-Lin algorithm to the task graph. Show the swap sequences, actually selected partitions and communication costs in each iteration of the algorithm.

(b) Can this algorithm still get stuck in a local minimum? Why or why not?

(c) How could this algorithm be extended to take computation costs and scheduling into account. Assume that tasks are periodic and that communication costs do not represent actual tasks dependencies (precedence constraints), but rather generally the fact whether two tasks ever exchange data or not (required connections).

(d) How could this algorithm be extended to support general *n*-way partitioning, i.e. more than two partitions?