Parallel Computer Architecture
Fall 2018

Homework 2

Tuesday, 11/12/2018

1) Modulo Scheduling (5+8+8+5+4 = 30 pt)

Consider the following for-loop and its translation in assembly. Assume that the code is executed in a wide-issue processor with 2 integer ALUs, 2 memory units, 1 FP adder, 1 FP multiplier, and 1 branch unit. The latencies of each instruction are the following: integer add 1 cycle, load 5 cycles, store 1 cycle, FP add 2 cycles, FP multiplies 4 cycles, branches 2 cycles. The purpose of this problem is to create a modulo schedule using the swing modulo schedule algorithm. You should study the paper on SMS by Josep Llosa et. al. before you attempt this exercise.

```
/* All operations are in FP */
for (i = 0; i < N; i++) {
    X = C[i] * B[i];
   A[i+3] = A[i] + X;
   B[i+3] = B[i] + A[i+3];
}
```

```
L1   t01 = load(tC);
L2   t02 = load(tB);
M1   tX = fmul(t01,t02);
L3   t03 = load(tA);
A1   t04 = fadd(t03,tX);
S1   store(tA3, t04);
A2   t05 = fadd(t02,t04);
S2   store(tB3, t05);
P1   tA = iadd(tA, #4);
P2   tB = iadd(tB, #4);
P3   tC = iadd(tC, #4);
P4   tA3 = iadd(tA3, #4);
P5   tB3 = iadd(tB3, #4);
P6   tI = iadd(iI, #1)
B1   bne tI, #100, top
```

a) As a first step, you will need to build the CDFG of this code including the edge annotation, i.e. the latency λ and the distance function δ of each edge. Make sure to include edges that may not be obvious from the assembly code. For example, there is a data dependence between the last statement of iteration i and the first statement of iteration i+3.

b) Compute the minimum possible value of the iteration interval when a compiler attempts to create a modulo schedule.

c) Use the SMS algorithm to create a modulo schedule based on the minimum II you computed in step (b). The SMS algorithm requires that you first compute the order in which the nodes are scheduled, and then to schedule the nodes. We ask for the steady state of the schedule in this question.

d) Finally, create the complete schedule for all possible number of iterations. Include the prologue and the epilogue on top of the steady state. Make sure that you unroll the kernel the appropriate number of times so that you avoid the problem of overlapping variable lives.

e) Tomasulo's algorithm provides an implicit register renaming mechanism by using Reorder Buffer slot

numbers as extra registers. Is it possible to use Tomasulo's algorithm (and the necessary hardware mechanisms) instead of kernel unrolling to perform register renaming in modulo schedule? Please, justify your answer.

2-3) Hardware/Software interaction. Exercises 3.1 and 3.2 from H&P, v.4. pg. 185-192 **(15+15pt)**

4) Hyperblocks (20pt)

The focus of this problem is to use compiler transformations such as predication and hyperblock formation to a relatively complex piece of source code in order to generate optimized machine code for an EPIC processor. The pseudo-code is given in the following table. For the purpose of this problem assume that each line of code takes 1 execution cycle, and translates to one assembly instruction. Statements such as SA1 are simple instructions (not conditionals).

```
for (i = 0; i < N; i++) {
    SA1;
    SA2;
    SA3;
    If (C11 || C12) then {
        SB11;
        SB2;
    }
    else
        SC1;
    SD1;
    If C2 exit;
    SE1;
    SE2;
    if ((C31 && C32) || C33) then {
        SF1;
        if (C4) then {
            SH1;
            if (C51 || C52 || C53) then {
                SK1;
            }
        }   // C4
        else {
            SG1;
            SG2;
            if (C6) then {
                SI1;
            }
            else {
                SJ1;
                SJ2;
                if  (C71 && ~C72) then {
                    SL1;
                }
            }
            SM1;
            SM2;
        }  // C4
    } // S3
    SN1;
```

} // for loop

a) Build the Control Flow Graph (CFG) of the code. The CFG describes only the control dependencies of the code, which is what we are concerned with when we apply predication. Nodes in the CFG are basic blocks and edges between two nodes show potential flow of control. You may merge statements that belong to the same BB into a single node, when you draw the graph. For example, instructions SA1, SA2, SA3 belong to the same BB.

b) Assume that profiling results have shown that most branches are not heavily biased towards any direction. This is a good test case for predication because predication mixes instructions from multiple paths and schedules the instructions together. Our block selection algorithm has determined that all statements except SC1 will be included in the hyperblock. Draw the hyperblock graph and make sure to apply *tail duplication* to accommodate the requirement that hyperblocks have a single entry at the top, and one or more exits. In other words, once the flow of control executes SC1, it cannot return back to the hyperblock.

c) The next step is to apply if-conversion to generate predicated code. Algorithms to do that focus on the structure of the CFG, and they determine first how many predicate variables are needed and where they should be placed in the code. Some definitions first:

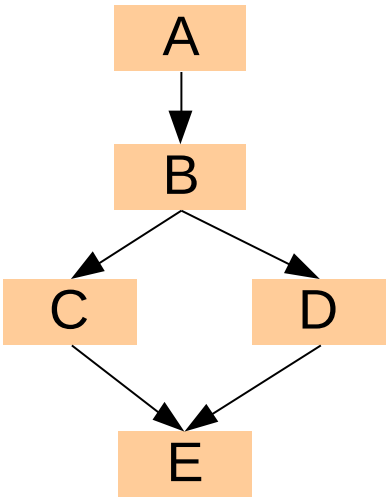
i) A node W is said to post-dominate a node V if every path from V to the end of the CFG contains W. By default, each node post-dominates itself.

ii) A node W is said to be control dependent on edge (U-->V) if:

W post-dominates V, and

if W is not U, then W does not post-dominate U.

For example, in the following CFG, node B post-dominates node A, but node C does not post-dominate A



(or B). Node C is control dependent to edge B-->C, but node E is not control dependent to edge B-->C.

The next step of the algorithm is to compute the control dependence set (CDS) of each node of the hyperblock. This set includes all the control dependent edges of the node. Each unique non-empty CDS requires a seperate predicate variable. For example, CDS(A) = {}, CDS(B) = {},CDS(C) = {B-->C}, CDS(D) = {B-->D}, and CDS(E) = {}. Since there are two unique non-empty CDS sets, there will be two predicates, p1 and p2:

p1 : B-->C

p2: B-->D

All the statements in BB C will be guarded by predicate p1, and all statements in BB D will be guarded by predicate p2. Statements in A, B, E will not have any predicates.

Apply if-conversion in the hyperblock of question (b), and write the predicated pseudo-code. You do not need to

worry about the tail-duplicated code, since the tail-duplicated code is not predicated in our system. Use the predicate arithmetic we introduced in the class to accurately specify the conditions under which one predicate becomes true or false.

d) Produce a schedule of the predicated version of the code. Assume an infinite number of resources, and assume that statements within a basic block (such as SA1, SA2, and SA3) cannot be scheduled in the same cycle due to data dependences. What is the speed up that you attain compared to the initial code?

5) Performance Estimation for Vector Processors and GPUs (10 pts)

In this problem, we will compare the performance of a 4-lane vector processor with a hybrid system that contains a scalar processor and a GPU-based coprocessor. Both processors are clocked at 1 GHz. The vector processor processes 4 data elements in a cycle which corresponds to 4 simple ops per cycle. In the hybrid system, the host processor has superior scalar performance to the GPU, so in this case all scalar code is executed in the host processor while all vector code is executed on the GPU. The GPU contains 64 streaming processors (SP) that can execute up to 64 CUDA threads simultaneously. We will refer to the first system as the vector computer and the second system as hybrid computer. Assume that your target application contains a vector kernel with an arithmetic intensity of 0.5 FLOPS per DRAM byte accessed; however the application also has a scalar component which must be executed before and after the kernel in order to prepare the input vectors and output vectors, respectively. For a sample dataset, the scalar portion of the code requires 400 ms of execution time on both the vector processor and the host processor in the hybrid system. The kernel reads input vectors consisting of 200 MB of data and has output data consisting of 100 MB of data. The vector processor has peak memory bandwidth of 30 GB/sec and the GPU has peak memory bandwidth of 150 GB/sec. The hybrid system has an additional overhead that requires all input vectors to be transferred between the host memory and the GPU local memory before and after the kernel is invoked. The hybrid system has a direct memory access (DMA) bandwidth of 10 Gb/sec and an average latency of 10 ms. Assume that both the vector processor and GPU are performance bound by memory bandwidth. Compute the execution time required by both computers for this application in the worst case, i.e. when the I/O does not overlap with computation, and in the best case, i.e. when I/O can fully overlap with computation.

6) Performance Estimation for GPUs (10 pts)

Assume a GPU architecture that contains 10 Streaming Multiprocessors (SMs), and 8 Streaming Processors (SPs) per SM. The warp size is 32 as in all NVIDIA GPUs, meaning that each non-diverged 32-thread warp can produce 32 results every 4 cycles. Assume a kernel that has divergent branches that cause on average 80% of threads to be active. Assume that 70% of all instructions executed are single-precision arithmetic and 20% are load/store. Since not all memory latencies are covered, assume an average warp-issue rate of 0.85. Assume that the GPU has a clock speed of 1.5GHz.

a) Compute the throughput in GFLOP/sec for this kernel on this GPU.

b) Assume that you have the following choices:

   1. Increasing the number of SPs to 16 per SM
   2. Increasing the number of SMs to 15 (assume that this change does not affect any other performance metrics and that the code scales to the additional processors)
   3. Adding a cache that effectively reduces memory latency by 40%, which will increase instruction issue rate to 0.95

   Which is bandwidth improvement in throughput for each of these improvements?