HY425

Homework Problem Set 3

Assignment: 9/4/2008

Due Date: 18/4/2008

Instructions: Solve all problems in a **.pdf** file and send them via e-mail to Stamatis Kavadias (kavadias@ics.forth.gr), with a copy to the instructor (dsn@ics.forth.gr). Use the following subject in your e-mail: **HY425: Homework 3 Submission**. Please the aforementioned subject only, so that your homework is read and graded.

Problem 1 (40 points)

Loops are often dominating the execution time of applications. Vector processors and multithreaded/multi-core processors can accelerate the execution of loops via vectorization or parallelization. The key idea is to distribute the iterations of a loop between threads, cores, or vector execution lanes. In order for this distribution to be valid, either the compiler, or the user, needs to prove that the iterations of a loop are independent. In particular, the compiler or the user needs to prove that there is no loop-carried dependence between loop iterations. More specifically: The iterations of a loop are "lexicographically" ordered. The lexicographical order corresponds to the order of execution of the loop iterations on a sequential processor. Generally speaking, a loop carried dependence exists if a loop iteration **`I`** which lexicographically precedes another loop iteration **`J`**, produces a result which is used (read or written) by iteration **`J`**. Note that this generalizes to nested loops, therefore **`I`** and **`J`** may actually be linear combinations of the actual loop indices (for example **`I=2×i+j`**, **`J=2×(i+1)+k`**, where **`i`**, **`j`**, and **`k`** are positive integers) The theory of data dependence analysis attempts to prove the existence or non-existence of dependences in loops, thereby enabling parallelization or vectorization.

You are given the following loop in C:

```
for (i=2;i<n;i++)
      for (j=2;j<i;j++)
            a[i,j]=a[i,j-2]*a[i,j-1]+b[i,j];
```

a. Show the loop-carried dependences for the aforementioned loop.
b. Rewrite the loop so that it can be vectorize. You don't need to actually provide the vector implementation in assembly, but you must show how the C code needs to be transformed for enabling vectorization.
c. The code that you derive in b. has a performance limitation, if implemented in a vector processor with an interleaved memory system. Find what it is, describe it in detail and propose a solution.

Problem 2 (30 points)

a. Vectorize the following loop, using exactly two vector instructions. You may use the VMIPS instruction set, described in detail in Appendix F of your textbook, or use a C-like notation for the vector operations, for example `A[0:10]=B[11:20]`,

```
for (i=1;i<n;i++)
   a[i] = b;
   c[i] = a[i-1];
```

Problem 3 (30 points)

Here is another unusual loop in C.

```
for (i=1;i<100;i++) {
   a[i] = b[i] + c[i];
   b[i] = a[i] + d[[i];
   a[i+1] = a[i] + e[i];
}
```

a. Rewrite the loop so that the loop can be perfectly vectorized.
b. Assume a vector processor which supports chaining. The processor users 4-element vectors and has 5-cycle pipelined execution lanes, organized in 2 lanes, where each lane has 2 vector add units. Assume the processor has a perfect memory system, which delivers vector elements with 0 latency. Plot the execution of the sequence above, after vectorization, and calculate how many cycles the processor needs to complete the sequence.