CORRECTED Handout 17

COMP 264: Introduction to Computer Systems (Section 001) Fall 2015 R. I. Greenberg
Comp. Sci. Dept., Loyola U., 820 N. Michigan Ave., Chicago, IL

Assignment #7

Issued 11/18; Corrected 11/23
Due 12/2

Best to hand in all solutions online. No fancy diagramming is required; you can scan something hand-drawn, or use any simple online tool of your choosing. Just make sure you end up with a common format that will be easy to handle, for example .pdf, .jpg, .txt or even .doc or .docx if you must. Also make sure that your submission name is appropriate to the file type, e.g., `7.pdf` or `7.txt`, etc.

With an appropriate extension as discussed above replacing "EXT" below: Homework is due at the time of day that class starts. Two options to submit your .EXT file:

- Easiest if you are working on a Loyola GNU/Linux machine: Copy your file to the directory `~rig/c264hw7sub` with a filename in the form `Email-X.EXT`, where `Email` is your email address, and `X` is a "random" string of at least 8 alphanumeric characters. The Unix command for this would look *similar* to:

  ```
  cp hw7.EXT ~rig/c264hw7sub/YOUREMAILADDRESS-RANDOM.EXT
  ```

  where you must put your own things for "YOUREMAILADDRESS" and "RANDOM". (Don't cut and paste from the PDF, or your tilde might not come out right.) *Remember that if you submit this way the file must be readable by everybody, though you will want to have used* `chmod` *to protect the directory containing the file. Protections show with the* `ls -l` *command illustrated below.* You can verify successful submision by using the "ls" command with the same file name you just copied to, specifically you can use a command *similar* to:

  ```
  ls -l ~rig/c264hw7sub/YOUREMAILADDRESS-RANDOM.EXT
  ```

- Or if you prefer: Submit the file through the online submission mechanism on my course web page. Submit it as `hw7.EXT` or `7.EXT`.

HW7-1 (43 points) Suppose we wish to write a procedure that computes the inner product of two vectors u and v. An abstract version of the function has a CPE of 14–18 with x86-64 for different types of integer and floating-point data. By doing the same sort of transformations we did to transform the abstract program `combine1` into the more efficient `combine4`, we get the following code:

```
void inner4(vec_ptr u, vec_ptr v, data t *dest) {
  long i;
  long length = vec_length(u);
  data_t *udata = get_vec_start(u);
```

```
  data_t *vdata = get_vec_start(v);
  data_t sum = (data_t) 0;

  for (i = 0; i < length; i++){
    sum = sum + udata[i] * vdata[i];
  }

  *dest = sum;
}
```

Our measurements show that this function has a CPE of 1.50 for integer data and 3.00 for floating-point data. For data type `double`, the x86-64 assembly code for the inner loop is as follows:

```
# Inner loop of inner4.  data_t = double.  OP = *.
# udata in %rbp, vdata %rax, sum in %xmm0, i in rcx, limit in rbx
.L15:                                      # loop:
        vmovsd 0(%rbp,%rcx,8), %xmm1       # Get udata[i]
        vmulsd (%rax,%rcx,8), %xmm1, %xmm1 # Multiply by vdata[i]
        vaddsd %xmm1, %xmm0, %xmm0         # Add to sum
        addq   $1, %rcx                    # Increment i
        cmpq   %rbx, %rcx                  # Compare i:limit
        jl     .L15                        # If <, goto loop
```

Assume that the functional units have the latencies and issue times given in Figure 5.12 (and in the course notes).

A. Diagram how this instruction sequence would be decoded into operations, and show how the data dependencies between them would create a critical path of operations in the style of Figures 5.13 (*Figure: opt/dpb-sequential*) and 5.14 (*Figure: opt/dpb-flow* and *Figure: opt/dpb-flow-abstract*). (25 points.)

B. For data type `double`, what lower bound on the CPE is determined by the critical path? (6 points.)

C. Assuming similar instruction sequences for the integer code as well, what lower bound on the CPE is determined by the critical path for integer data? (6 points.)

D. Explain how the floating-point version can have a CPE of 3.00 even though the multiplication operation requires 5 cycles. (6 points.)

HW7-2 (27 points) Write a version of the inner product procedure described in the previous problem that uses six-way loop unrolling (6×1; no parallelism). (11 points.)

For x86-64, our measurements of the unrolled version give a CPE of 1.07 for integer data but still 3.01 for floating-point data.

A. Explain why any version of any inner product procedure cannot achieve a CPE less than 1.00. (8 points.)

B. Explain why the performance for floating-point data did not improve with loop unrolling. (8 points.)