Lab Exam

Computer Architecture - CS2323. Autumn 2024

Time: 2.5 hours

23-Nov-2024

ACCESS TO INTERNET/DOWNLOADED CODE DURING THE EXAM IS STRICTLY NOT ALLOWED

Part-1: Assembly code for matrix multiplication

We need to multiply two matrices (A and B) and store the result matrix (C) in a different location in memory. The matrix elements are 64-bit **signed** integers (dwords). The location starting from 0x1000_0000 (beginning of .data section) stores the size of two matrices in this order: rowA, colA, rowB, colB. After this, the elements of matrix-A are stored and then, the elements of matrix-B are stored. Both matrices are stored in row-major order.

The result is to be stored in memory starting from location 0x1000_4000 (data segment base + 0x4000). The first two locations should contain the size of the result matrix in row, col order and then the matrix elements in row-major order.

You can use any instruction from the base class of RISC-V, including any pseudo instructions for your implementation. Use of M extension (mul instruction) is also allowed.
The matrix multiplication algorithm should be implemented as a function named (mmult) which:

a. Takes dimensions of A and B matrices as well as the starting address of A, B, C as inputs.
b. Performs the matrix multiplication and stores the result in the location of C.
c. Performs error handling if the matrix dimensions are not compatible.
d. Follows caller-callee conventions.

A pseudo-code for matrix multiplication in C syntax is given below and you should implement the same flow, without any optimizations:

```
for (int i = 0; i < rowA; i++) {
    for (int j = 0; j < colB; j++) {
        C[i][j] = 0;
        for (int k = 0; k < rowB; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

A template for the assembly code file is as written below:

```
.data
.dword rowA, colA, rowB, colB, A00, A01, A02..., A10, A11, ..., B00, B01,... B10, B11,...

.text
  #your code starts here
  #The final result should be in memory starting from address 0x1000_4000
  #The first and second dword location at 0x1000_4000 contains rowC, colC
  #The third location from 0x1000_4000 contains the result matrix C00, C01, ...C10,C11, ...
```

Example: If the data section contains .dword **3, 4, 4, 2,** 3, 1, 2, 4, 1, 1, 1, 2, 2, 3, 1, 1, 1, 2, 2, 2, 3, 4, 4, 1
It means we have 3x4 and 4x2 matrices to be multiplied. The matrix values are also given. After executing your code, memory location from 0x1000_4000 should contain dwords like **3, 2,** 27, 20, 14, 10, 15, 15.

Assumptions:

1. Assume that the multiplication operation will NOT cause any overflow.
2. The matrix sizes will be such that they fit in the region between 0x1000_0000 to 0x1000_4000
3. The dimensions of C matrix is written as 0, 0 in case of any erroneous inputs

Coding style:

1. The code should be properly indented
2. The code should follow RISC-V convention (use add x2, x3, x4 instead of add x2 x3 x4)
3. The code should include inline comments for each major block indicating what is being done in that block

---

Part-2: Cache analysis for Part-1: Will be considered only if Part-1 is complete and working well.

Assume a data cache of size 64 Bytes, with 16 Bytes block size, and associativity of 2. Write policy is write-back with allocate and replacement policy is LRU.
Modify your assembly implementation compared to the original flow given in the pseudo-code so that the number of cache misses reduces significantly. Please note that the modified code should compute matrix multiplication correctly. Mention the original and improved number of cache hit and miss when matrix A and B are both of dimensions:

a. 8x8 and 8x8
b. 16x32 and 32x~~64~~ 16
c. ~~64x16 and 16x32~~ 16x16 & 16x16
d. ~~64x64 and 64x64~~ 32x16 & 16x32

Submission instructions:

1. Create and submit a zip file on moodle on the link corresponding to the Lab Exam
2. File should be named LABEXAM_ROLLNUM.zip (LABEXAM_CSYYBTECHXXXXX.zip)
3. It should contain a report.pdf and the assembly code (main_part1.s and main_part2.s)
4. report.pdf should contain your implementation details and inputs used to verify the code. If you are doing part-2 also, the report should also include your approach to improving the cache performance.
5. The code should be working in RIPES simulator (64-bit RISC-V CPU) without any changes
6. The input and output format/register should exactly match as specified
7. You can use code from your previous submissions, if needed

---

ACCESS TO INTERNET/DOWNLOADED CODE DURING THE EXAM IS STRICTLY NOT ALLOWED
