IB Computer Design - Supervision 5

Nandor Licker <nl364@cl.cam.ac.uk>

Due noon before the day of supervision

Q1 Consider the following matrix multiplication implementation:

```
void tiled_multiply(const float *a, const float *b, float *c)
{
  for (int bj = 0; bj < N; bj += T) {
    for (int bk = 0; bk < N; bk += T) {
      for (int i = 0; i < N; ++i) {
        for (int j = bj; j < bj + T; ++j) {
          float r = 0.0;
          for (int k = bk; k < bk + T; ++k) {
            r += a[i * N + k] * b[k * N + j];
          }
          c[i * N + j] += r;
        }
      }
    }
  }
}
```

Assuming a block size of 8, provide an implementation using the AVX-2 instruction set extension in C. You should use intrinsic functions, enabled using the `-mavx2` flag passed to the compiler and exposed in the `<immintrin.h>` header. You may find the following intrinsics useful:

- `_mm256_castps256_ps128`
- `_mm256_dp_ps`
- `_mm256_i32gather_ps`
- `_mm256_load_ps`
- `_mm_add_ss`
- `_mm_load_ss`
- `_mm_store_ss`

Q2 Answer the following questions about vectorised instructions:

1. What are the challenges of implementing gather/scatter instructions? Think about cache lines. Implement the 8-lane gather instruction with other SSE instructions, without using gather. Why is it advantageous to have a single instruction doing that much work?
2. Why doesn't the 8-lane dot product actually compute a dot product?
3. Show how you would implement a masked instruction from the AVX-512 instruction set using only AVX-2 instructions. Comment on the advantages of masks. Comment on how masks can reduce power usage in an actual hardware implementation.
4. Transpose a N×N matrix using scatter instructions. What is the bottleneck of a naive implementation? How does tiling help? Compare the number of bytes written to main memory.
5. How effective is hyperthreading when both thread run vector-heavy code? Could you transpose two matrices on two threads in the same time as one matrix on one thread?

Q3 Consider the following RISC-V instruction sequence:

```
lw t0, 10(r0)
addi t0, t0, 1
sw t0, 10(r0)
```

Answer the following questions:

1. Draw a timing diagram indicating how this instruction would execute on a typical 5-stage pipeline. Consider the `load`, `store` and `add` instructions. Describe what happens during the execution of each instruction in each of the pipeline stages, enumerating the actions in a list.
2. Show how you would reduce the pipeline to 4 stages. What stages would you keep and what new stage would you introduce? Show how the instructions would execute on this pipeline.
3. What is the main disadvantage of the previous implementation? Increase the maximum clock rate by providing a 5-stage implementation which does not stall. Describe the function of each pipeline stage.

   Hint: Consider `lw t0, 10(r0)`. Compute `r0 + 10` in one stage and do the load in another. In which stage do you need to perform the addition for the `addi` instruction?
4. Describe in detail at least 3 different ways of forwarding `t0` to the last add instruction.

   ```
   addi t0, t0, 1
   addi t1, t0, 1
   addi t2, t1, t0
   ```