Programming Parallel Computers

Chapter 2: Case study

Version 4: Assembly code [advanced]

Let us first check that the compiler indeed did what we wanted. Our plan was that we would read 3 + 3 vectors, do 3 × 3 pairwise vector additions, and then update 3 × 3 minimums. This is exactly what the compiler gave us:

```
LOOP:
        leaq        (%rax,%rdi), %rcx
        vmovaps     (%rax), %ymm2
        addq        $32, %rax
        vmovaps     (%rdx), %ymm3
        vmovaps     (%rcx,%r9), %ymm1
        vmovaps     (%rcx,%rsi), %ymm0
        leaq        (%rdx,%r10), %rcx
        addq        $32, %rdx
        cmpq        %r8, %rax
        vmovaps     (%rcx,%rbx), %ymm14
        vmovaps     (%rcx,%r11), %ymm13
        vaddps      %ymm14, %ymm2, %ymm15
        vminps      %ymm15, %ymm12, %ymm12
        vaddps      %ymm14, %ymm1, %ymm15
        vaddps      %ymm14, %ymm0, %ymm14
        vminps      %ymm15, %ymm11, %ymm11
        vminps      %ymm14, %ymm10, %ymm10
        vaddps      %ymm3, %ymm2, %ymm14
        vaddps      %ymm13, %ymm2, %ymm2
        vminps      %ymm14, %ymm9, %ymm9
        vaddps      %ymm3, %ymm1, %ymm14
        vaddps      %ymm3, %ymm0, %ymm3
        vaddps      %ymm13, %ymm1, %ymm1
        vaddps      %ymm13, %ymm0, %ymm0
        vminps      %ymm14, %ymm8, %ymm8
        vminps      %ymm3, %ymm7, %ymm7
        vminps      %ymm2, %ymm6, %ymm6
        vminps      %ymm1, %ymm5, %ymm5
        vminps      %ymm0, %ymm4, %ymm4
        jne         LOOP
```

We can count 6 vector reads from the memory (`vmovaps`), 9 vector additions (`vaddps`), and 9 vector minimums (`vminps`). All intermediate results are kept in the vector registers (`%ymm`). The only memory accesses in the innermost loop are reads.

It is also good to note that this code is using as many as **16 vector registers**:

- 6 registers for the values that we read from the memory (registers 0–3, 13, 14),
- 9 register for the minimums that we accumulate (registers 4–12),
- 1 register for temporary values (register 15).

There is a little bit of room for saving some registers, but no matter what we do, we will need at least 9 registers for the minimums that we accumulate, plus some number of registers for the values that we read and want to keep around for reuse.

The CPU that we use has only got 16 vector registers. Hence, in a sense the scheme that we used cannot be improved much further. If we tried to calculate a 4 × 4 block of the results by scanning 4 rows and 4 columns, we would run out of registers.

Interactive assembly

You can use **Compiler Explorer** to try it out:

- Try to extend the block size to e.g. 4 × 4; what happens to the assembly code?
- What changes if your try 4 × 4 but set the target architecture to `-march=cascadelake` ?