A3

Alexander Mueller and Jonas Bonnaudet

November 2022

1 NUMA

1.1 Performance difference

1.1.1 Deöptimized performance

Our deöptimized code sees only slight differences in performance. We see that local allocation has the best performance, at 1.422e-08, then interleaved at 1.684e-08, and finally remote at 2.005e-08 seconds per access. This is expected because processors accessing local memory is supposed to be faster than accessing remote memory.

1.1.2 Optimized

The optimized code runs too fast to measure reasonable time differences no matter what policy is used. The time per access is around 3.08e-10 seconds which shows that the cache prediction is doing a very good job.

1.2 Deöptimization

In order to deöptimize the given code, we assigned a random value to each position in the array from the possible values {8, 16, 24, 32, 40, 48, 56, 64}. This has no effect on performance. Then, for next_addr, we simply return the value in the array at position i. This guarantees that we are not in the same cache block anymore since we are at least 8 elements away from the current element and that we aren't jumping linearly through the array, since all next accesses are completely random. That way the stride prefetcher is less effective.

2 Order

2.1 Assembly analysis

The instructions that are reordered are the ones that set X (or Y) to 1 and the one that puts Y (or X) to X (or Y).

```
movl    $1, X(%rip)        # set X to 1
movl    Y(%rip), %eax      # set eax to Y
movl    %eax, r1(%rip)     # set r1 to eax (Y in this case)
```

and

```
movl    $1, Y(%rip)        # set Y to 1
movl    X(%rip), %eax      # set eax to X
movl    %eax, r2(%rip)     # set r2 to eax (X in this case)
```

2.2 Disabling reöderings

To disable reöderings, we used

```
asm volatile("mfence" ::: "memory");
```

instead of

```
asm volatile("" ::: "memory");
```

2.3 Hardware behavior

On a fence, the CPU blocks until the SB is empty, guaranteeing serializability.

2.4 Reördering across sockets

In our testing, we find that running the explicit threads on the same socket leads to more than 90% of memory instructions being reordered. In contrast, running the explicit threads on separate threads drastically reduces the amount of reördering: we only saw 45% of memory instructions being reördered. This is because the amount of interleaving is much lower when the threads access values in remote memory, causing less reördering overall.