Parallel & Vectorized Execution

This lesson explains parallel and vectorized execution policies, which were introduced in C++ 17, in detail.

WE'LL COVER THE FOLLOWING

Parallel & Vectorized Execution

Whether an algorithm runs in a parallel and vectorized way depends on many factors. For example, it depends on whether the CPU and the operating system support SIMD instructions. Additionally, it also depends on the compiler and the optimization level that you used to translate your code.

The following example shows a simple loop for creating a new vector.

```
const int SIZE= 8;

int vec[] = {1, 2, 3, 4, 5, 6, 7, 8};
int res[] = {0, 0, 0, 0, 0, 0, 0, 0};
int main(){
   for (int i= 0; i < SIZE; ++i) {
     res[i]= vec[i]+5;
   }
}
```

Line 8 is the key line in this small example. Thanks to the compiler explorer, we can have a closer look at the assembler instructions generated by clang 3.6.

Without Optimisation?

Here are the assembler instructions. Each addition is done sequentially.

```
movslq  -8(%rbp), %rax
movl    vec(,%rax,4), %ecx
addl    $5, %ecx
movslq  -8(%rbp), %rax
movl    %ecx, res(,%rax,4)
```

With maximum Optimization?

By using the highest optimization level, `-O3`, special registers such as `xmm0` are used that can hold 128 bits or 4 `int`s. This means that the addition takes place in parallel on four elements of the vector.

```
movdqa  .LCPI0_0(%rip), %xmm0   # xmm0 = [5,5,5,5]
movdqa  vec(%rip), %xmm1
paddd   %xmm0, %xmm1
movdqa  %xmm1, res(%rip)
paddd   vec+16(%rip), %xmm0
movdqa  %xmm0, res+16(%rip)
xorl    %eax, %eax
```

Hazards of Data Races and Deadlocks

The parallel algorithm does not automatically protect you from data races and deadlocks.

```
int numComp= 0;

std::vector<int> vec={1,3,8,9,10};

std::sort(std::parallel::vec, vec.begin(), vec.end(),
          [&numComp](int fir, int sec){ numComp++; return fir < sec
; });
```

The small code snippet has a data race. `numComp` counts the number of operations, which means that `numComp` in particular is concurrently modified in the lambda-function. In order to be well-defined, `numComp` has to be protected.