

Scope of variables

How does the scope of a variable work in openMPI?

To summarize, then, variables that have been declared before a `parallel` directive have shared scope among the threads in the team, while variables declared in the block (e.g., local variables in functions) have private scope. Furthermore, the value of a shared variable at the beginning of the `parallel` block is the same as the value before the block, and, after completion of the `parallel` block, the value of the variable is the value at the end of the block.

We'll shortly see that the *default* scope of a variable can change with other directives, and that OpenMP provides clauses to modify the default scope.

- One of the important things to note is that a variable that is shared will be affected in the parallel function if it is changed.

Full section if I want to read it:

- It goes over it with examples from the trapezoidal program

In serial programming, the *scope* of a variable consists of those parts of a program in which the variable can be used. For example, a variable declared at the beginning of a C function has "function-wide" scope, that is, it can only be accessed in the body of the function. On the other hand, a variable declared at the beginning of a .c file but outside any function has "file-wide" scope, that is, any function in the file in which the variable is declared can access the variable. In OpenMP, the **scope** of a variable refers to the set of threads that can access the variable in a `parallel` block. A variable that can be accessed by all the threads in the team has **shared** scope, while a variable that can only be accessed by a single thread has **private** scope.

In the "hello, world" program, the variables used by each thread (`my_rank` and `thread_count`) were declared in the `Hello` function, which is called inside the `parallel` block. Consequently, the variables used by each thread are allocated from the thread's (private) stack, and hence all of the variables have private scope. This is *almost* the case in the trapezoidal rule program; since the `parallel` block is just a function call, all of the variables used by each thread in the `Trap` function are allocated from the thread's stack.

However, the variables that are declared in the `main` function (`a`, `b`, `n`, `global_result`, and `thread_count`) are all accessible to all the threads in the team started by the `parallel` directive. Hence, the *default* scope for variables declared before a `parallel` block is shared. In fact, we've made implicit use of this: each thread in the team gets the values of `a`, `b`, and `n` from the call to `Trap`. Since this call takes place in the `parallel` block, it's essential that each thread has access to `a`, `b`, and `n` when their values are copied into the corresponding formal arguments.

Furthermore, in the `Trap` function, although `global_result_p` is a private variable, it refers to the variable `global_result` which was declared in `main` before the `parallel` directive, and the value of `global_result` is used to store the result that's printed out after the `parallel` block. Thus in the code

```
*global_result_p += my_result;
```

it's essential that `*global_result_p` have shared scope. If it were private to each thread, there would be no need for the `critical` directive. Furthermore, if it were private, we would have a hard time determining the value of `global_result` in `main` after completion of the `parallel` block.

To summarize, then, variables that have been declared before a `parallel` directive have shared scope among the threads in the team, while variables declared in the block (e.g., local variables in functions) have private scope. Furthermore, the value of a shared variable at the beginning of the `parallel` block is the same as the value before the block, and, after completion of the `parallel` block, the value of the variable is the value at the end of the block.

We'll shortly see that the *default* scope of a variable can change with other directives, and that OpenMP provides clauses to modify the default scope.