< Table Of Contents

omp declare reduction

Declares User-Defined Reduction (UDR) functions (reduction identifiers) that can be used as reduction operators in a reduction clause.

Syntax

```
#pragma omp declare reduction (reduction-identifier : typename-list :
combiner) [initializer-clause]
```

Arguments

reduction-identifier	Is either an identifier or one of the following operators: +, -, *, &, |, ^, && and ||
typename-list	Is a list of type names.
combiner	Is an expression that specifies how partial results can be combined into a single value.
initializer-clause	Is `initializer` (*initializer-expr*) where *initializer-expr* is `omp_priv` = *initializer* or *function-name* (*argument-list*). At most one *initializer-clause* can be specified. If no *initializer-clause* is specified, the private variables will be initialized following the C rules for initialization of objects with static storage duration. If no *initializer-expr* is specified, the private variables will be initialized following the C++ rules for default initialization.

Description

When defining custom reductions, the *reduction-identifier* and the type identify the declare reduction pragma. The *reduction-identifier* can later be used in a reduction clause using variables of the type or types specified in the `declare reduction` pragma. If the pragma applies to several types, it is treated as if there were multiple declare reduction pragmas, one for each type.

The visibility and accessibility of this declaration are the same as those of a variable declared at the same point in the program. The enclosing context of the *combiner* and of the *initializer-expr* (if specified) is that of the declare reduction pragma. The *combiner* and the *initializer-expr* must be correct in C/C++ as if they were the body of a function defined at the same point in the program.

NOTE
A *reduction-identifier* cannot be re-declared in the current scope for the same type or for a type that is compatible according to C/C++ rules.

The following rules and restrictions apply to the *combiner*:

- The *combiner* can use the special variable identifiers `omp_in` and `omp_out` that are of the type of the variables being reduced with this *reduction-identifier*. These are the only variables allowed in the *combiner*.

- The `omp_in` identifier refers to the partial result local to each thread, while the `omp_out` identifier refers to the storage that holds the resulting combined value after executing the *combiner*.
- The number of times the *combiner* is executed, and the order of these executions, for any `reduction` clause is unspecified.

The following rules and restrictions apply to the optional *initializer-expr*:

- If *initializer-clause* is specified, the *initializer-expr* value is used as the initializer for private copies of reduction list items where the `omp_priv` identifier refers to the storage to be initialized.
- Only the variables `omp_priv` and `omp_orig` are allowed in the *initializer-clause*.
- The special identifier `omp_orig` refers to the storage of the original variable to be reduced. If `omp_orig` is modified in the *initializer-clause*, the behavior is unspecified.
- The number of times that the *initializer-expr* is evaluated, and the order of these evaluations, is unspecified.
- If the *initializer-expr* is a function name with an argument list:
  - One of the arguments must be `omp_priv` or the address of `omp_priv`.
  - The *initializer-expr* is evaluated by calling the function with the specified argument list.

    Otherwise, the C/C++ *initializer-expr* specifies how `omp_priv` is declared and initialized.

NOTE
If execution of the *combiner* or the *initializer-expr* results in the execution of an OpenMP construct or an OpenMP API call, then the behavior is unspecified.

Example

```
#define abs(x)    (x<0 ? -x : x)
#define LARGENUM 2147483647
#define N        1000000
int data[N];

// return the smallest magnitude among all the integers in data[N]
int find_min_abs()
{
  int i;
  int result = LARGENUM;

  #pragma omp declare reduction(minabs : int :              \
    omp_out = abs(omp_in) > omp_out ? omp_out : abs(omp_in) \
    initializer (omp_priv=LARGENUM)

  #pragma omp parallel for reduction(minabs:result)
  for (i=0; i<N; i++) {
    if (abs(data[i]) < result) {
      result = abs(data[i]);
    }
  }

  return result;
}
```

Parent topic: Intel-supported OpenMP* Pragma Reference