THE REDUCTION CLAUSE

- A*reduction* operator is a binary operation (such as addition or multiplication) anda reduction is a computation that repeatedly applies the same reduction operator toa sequence of operands in order to get a single result.
- All of the intermediateresults of the operation should be stored in the same variable: the reductionvariable. For example, if A is an array of ***n*** **ints**, the computation.

  ```
  int sum = 0;
  for (i = 0; i < n; i++)
  sum += A[i];
  ```

  is a reduction in which the reduction operator is addition

- In OpenMP it may be possible to specify that the result of a reduction is a reduction variable. To do this, a reduction clause can be added to a parallel directive.
- In our example, we can modify the code as follows:

  ```
  global_result = 0.0;
  # pragmaomp parallel num threads(thread count) \
  reduction(+: global result)
  global result += Local trap(double a, double b, int n);
  ```

- The syntax of the reduction clause is

  ```
  reduction(<operator>: <variable list>)
  ```

- In C, operator can be any one of the operators +,*,-, &, |, , ˆ, &&, ||, although the use of subtraction is a bit problematic, since subtraction isn't associative or commutative.

  For example, the serial code

  ```
  result = 0;
  for (i = 1; i <= 4; i++)
      result-= i;
  ```

  stores the value -10 in result.

- When a variable is included in a reduction clause, the variable itself is shared. However, a private variable is created for each thread in the team. In the ***parallel*** block each time a thread executes a statement involving the variable, it uses the private variable. When the ***parallel*** block ends, the values in the private variables are combined into the shared variable. Thus, our latest version of the code

  ```
  Global_result = 0.0;
  # pragmaomp parallel num_threads(thread_count) n
  reduction(+: global_result)
  global_result += Local_trap(double a, double b, int n);
  ```

  effectively executes code that is identical to our previous version:

  ```
  global_result = 0.0;
  # pragmaomp parallel num_threads(thread_count)
  {
  doublemy_result = 0.0; /_ private _/
  my_result += Local trap(double a, double b, int n);
  # pragmaomp critical
  Global_result += my_result;
  }
  ```