operator	initial value
+	0
-	0
*	1
&	bit string of 1s
|	bit string of 0s
^	0
&&	1
||	0

The lack of other operations typically found in other parallel programming languages, such as min and max, is due to the lack of these operators in C/C++. The FORTRAN version of OpenMP does have min and max.[3]

4.4 The Task Directive

This is new to OpenMP 3.0. The basic idea is this: When a thread encounters a **task** directive, it arranges for some thread to execute the associated block. The first thread can continue.

Here's a Quicksort example:

```
1  // OpenMP example program:  quicksort; not necessarily efficient
2
3  void swap(int *yi, int *yj)
4  {  int tmp = *yi;
5     *yi = *yj;
6     *yj = tmp;
7  }
8
9  int *separate(int *x, int low, int high)
10 {  int i,pivot,last;
11    pivot = x[low];  // would be better to take, e.g., median of 1st 3 elts
12    swap(x+low,x+high);
13    last = low;
14    for (i = low; i < high; i++) {
15       if (x[i] <= pivot) {
16          swap(x+last,x+i);
17          last += 1;
18       }
19    }
20    swap(x+last,x+high);
21    return last;
22 }
23
24 // quicksort of the array z, elements zstart through zend; set the
25 // latter to 0 and m-1 in first call, where m is the length of z;
26 // firstcall is 1 or 0, according to whether this is the first of the
27 // recursive calls
```

[3]Note, though, that plain min and max would not help in our Dijkstra example above, as we not only need to find the minimum value, but also need the vertex which attains that value.

```
28 void qs(int *z, int zstart, int zend, int firstcall)
29 {
30    #pragma omp parallel
31    {  int part;
32       if (firstcall == 1) {
33          #pragma omp single nowait
34          qs(z,0,zend,0);
35       } else {
36          if (zstart < zend) {
37             part = separate(z,zstart,zend);
38             #pragma omp task
39             qs(z,zstart,part-1,0);
40             #pragma omp task
41             qs(z,part+1,zend,0);
42          }
43
44       }
45    }
46 }
47
48 main(int argc, char**argv)
49 {  int i,n,*w;
50    n = atoi(argv[1]);
51    w = malloc(n*sizeof(int));
52    for (i = 0; i < n; i++) w[i] = rand();
53    qs(w,0,n-1,1);
54    if (n < 25)
55       for (i = 0; i < n; i++) printf("%d\n",w[i]);
56 }
```

The code

```
if (firstcall == 1) {
   #pragma omp single nowait
   qs(z,0,zend,0);
```

gets things going. We want only one thread to execute the root of the recursion tree, hence the need for the **single** clause. After that, the code

```
part = separate(z,zstart,zend);
#pragma omp task
qs(z,zstart,part-1,0);
```

sets up a call to a subtree, with the **task** directive stating, "OMP system, please make sure that this subtree is handled by some thread."

This really simplifies the programming. Compare this to the Python **multiprocessing** version in Section 3.5, where the programmer needed to write code to handle the work queue.

There are various refinements, such as the barrier-like **taskwait** clause.