A little more OpenMp...

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 10TB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW OpenMP
#pragma
TAP TAP

xkcd208

A little more OpenMp...

```
#pragma omp parallel
{
  int id = omp_get_thread_num();
  array[id] = some_calculation();
#pragma omp barrier
  array2[id] = array[id] + array[id + 1];
}
```

A little more OpenMp...

```
#pragma omp parallel
{
  int id = omp_get_thread_num();
  array[id] = some_calculation();
#pragma omp barrier
  array2[id] = array[id] + array[id + 1];
}
------------------------------------------------
#pragma omp parallel
{
#pragma omp for
  for (int id=0; id<number_of_threads; tid++)
    x[id] = some_calculation();
// implicit barrier
#pragma omp for
  for (int id=0; id<number_of_threads-1; id++)
    y[id] = x[id] + x[id + 1];
}
```

A little more OpenMp...

```
#pragma omp parallel
{
#pragma omp for nowait
  for (int id=0; id<number_of_threads; tid++)
    x[id] = some_calculation();
// no more implicit barrier!!!
#pragma omp for
  for (int id=0; id<number_of_threads-1; id++)
    y[id] = x[id] + x[id + 1];
}
```

A little more OpenMp...

<u>L O C K S ! ! !</u>

Create/destroy:

```
void omp_init_lock(omp_lock_t *lock);
void omp_destroy_lock(omp_lock_t *lock);
```

Set and release:

```
void omp_set_lock(omp_lock_t *lock);
void omp_unset_lock(omp_lock_t *lock);
```

Since the set call is blocking, there is also

```
omp_test_lock();
```

Unsetting a lock needs to be done by the thread that set it.

A little more OpenMp...

```
#pragma omp parallel for private(j) collapse(2)
for (i = 0; i < 3; i++)
    for (j = 0; j < 9; j++)
       printf("%d,%d :: %d \n", i, j, omp_get_thread_num());
```

collapse(2)

```
2,3 :: 3     0,0 :: 0
2,4 :: 3     0,1 :: 0
2,5 :: 3     0,2 :: 0
1,5 :: 2     0,3 :: 0
1,6 :: 2     0,4 :: 0
1,7 :: 2     0,5 :: 0
2,6 :: 3     0,6 :: 0
2,7 :: 3     0,7 :: 1
2,8 :: 3     0,8 :: 1
1,8 :: 2     1,0 :: 1
2,0 :: 2     1,1 :: 1
2,1 :: 2     1,2 :: 1
2,2 :: 2     1,3 :: 1
             1,4 :: 1
```

without collapse(2)

```
             1,3 :: 1
             1,4 :: 1
0,0 :: 0     1,5 :: 1
0,1 :: 0     1,6 :: 1
0,2 :: 0     1,7 :: 1
0,3 :: 0     1,8 :: 1
0,4 :: 0     2,0 :: 2
0,5 :: 0     2,1 :: 2
0,6 :: 0     2,2 :: 2
0,7 :: 0     2,3 :: 2
0,8 :: 0     2,4 :: 2
1,0 :: 1     2,5 :: 2
1,1 :: 1     2,6 :: 2
1,2 :: 1     2,7 :: 2
             2,8 :: 2
```

A little more OpenMp...

```
// any single thread should execute

#pragma omp single [clauses]
{
   code_block
}

// only master thread should execute

#pragma omp master
{
   code_block
}
```

A little more OpenMp...

```
// calling function in a galaxy far far away...

#pragma omp parallel
{
   // this block is parallel!!!
   code_block
}

// Orphaning
// your local code

#pragma omp parallel
{
   // this block is also parallel!!!
   code_block
}
```

A little more OpenMp...

```
// calling function in a galaxy far far away...

{
   // this block is NOT parallel!!!
   code_block
}

// Orphaning
// your local code
// this will run sequential

#pragma omp parallel
{
   // this block is also NOT parallel!!!
   code_block
}
```