OpenMP

OpenMP adds constructs for shared-memory threading to C/Fortran

```
for (i = 0; i < n; i++)
  array[i] = convert(array[i]);
```

⇒

```
#pragma omp parallel for
for (i = 0; i < n; i++)
  array[i] = convert(array[i]);
```

Compiling with OpenMP

Run **gcc** with the **-fopenmp** flag:

```
gcc -O2 -fopenmp ex1.c
```

Beware: If you forget **-fopenmp**, then all OpenMP directives are ignored!

Reflecting on Threads

Include **omp.h** to get extra functions:

```
#include <omp.h>
#include <stdio.h>

int main() {
  #pragma omp parallel
  printf("hello from %d of %d\n",
         omp_get_thread_num(),
         omp_get_num_threads());
}
```

Copy

Running OpenMP Programs

To control the number of threads used to run an OpenMP program, set the **OMP_NUM_THREADS** environment variable:

```
% ./a.out
hello from 0 of 2
hello from 1 of 2

% env OMP_NUM_THREADS=3 ./a.out
hello from 2 of 3
hello from 0 of 3
hello from 1 of 3
```

OpenMP Directives

For C, OpenMP directives start

```
#pragma omp
```

Some directives that can follow that prefix:

- `parallel`
  - `private`, `shared`, `default`
  - `reduction`
- `for`
- `sections`, `section`
- `barrier`
- `exclusive`

Creating Threads

The **`parallel`** directive creates threads and runs following statement/block in each thread

```
#pragma omp parallel
printf("hello");
```

Threads and Sharing

Variables outside a **`parallel`** are shared, and variables inside a **`parallel`** are private

`private`, **`shared`** and **`default`** control sharing:

```
#include <omp.h>
#include <stdio.h>

int main() {
  int t, j, i;
  #pragma omp parallel private(t, i) shared(j)
  {
    t = omp_get_thread_num();
    printf("running %d\n", t);
    for (i = 0; i < 1000000; i++)
      j++; /* race! */
    printf("ran %d\n", t);
  }
  printf("%d\n", j);
}
```

Copy

Reduce

The **`reduction`** clause of **`parallel`**

- makes the specified variable private to each thread
- combines private results on exit

```
int t;
#pragma omp parallel reduction(+:t)
{
  t = omp_get_thread_num() + 1;
  printf("local %d\n", t);
}
printf("reduction %d\n", t);
```

Copy

Work Sharing

With a parallel section, ***workshare*** directives split work among the available threads:

- `for`
- `sections`
- `single`

Unless the `nowait` clause is specified, each workshare is followed by an implicit barrier

Loop Workshare for Data Parallelism

The **for** workshare directive

- requires that the following statement is a **for** loop
- makes the loop index private to each thread
- runs a subset of iterations in each thread

```
#pragma omp parallel
#pragma omp for
for (i = 0; i < 5; i++)
  printf("hello from %d at %d\n",
         omp_get_thread_num(), i);
```

Copy

Or use **#pragma omp parallel for**

Combining Loop and Reduce

```
int array[8] = { 1, 1, 1, 1, 1, 1, 1, 1};
int sum = 0, i;
#pragma omp parallel for reduction(+:sum)
for (i = 0; i < 8; i++) {
  sum += array[i];
}
printf("total %d\n", sum);
```

Copy

Section Workshare for Task Parallelism

A **`sections`** workshare directive is followed by a block that has **`section`** directives, one per task

```
#pragma omp parallel
#pragma omp sections
{
  #pragma omp section
  printf("Task A: %d\n", omp_get_thread_num());
  #pragma omp section
  printf("Task B: %d\n", omp_get_thread_num());
  #pragma omp section
  printf("Task C: %d\n", omp_get_thread_num());
}
```

Copy

Other Patterns

When OpenMP doesn’t provide high-level support for your goal (e.g., there’s no **`scan`** directive), you can always fall back to manual data management and synchronization

Synchronization

- `barrier` within a parallel block is as in Peril-L
- `exclusive` within a parallel block is as in Peril-L
- `atomic` is a restricted form of `exclusive`

OpenMP Documentation

http://www.openmp.org/