

Linked list traversal

07/12/2016

1 Linked list traversal

This exercise is about parallelizing the traversal of a singly linked list whose length is unknown.

The list is formed by elements of this type

```
struct node{
    unsigned long val;
    struct node *next;
};
```

Where the `val` field contains an integer value and the `next` field is a pointer to the next element in the list. The list can be accessed through the `head` pointer which points to the first element in the list.

2 Package content


In the `linked_list` directory you will find the following files:

- `main.c`: this file contains the main program which first calls the `init_list` routine which initializes the list with a fixed, unknown length. The main program then calls a sequential routine `sequential_sweep` and then the two parallel routines `parallel_for_sweep` and `parallel_task_sweep`. Each of these three routines traverses the whole linked list and processes each element calling the `process_node` routine on it; the result of this processing is accumulated in the `acc` variable and returned at the end of the routine. The `process_node` operation is expensive and takes a relatively long time to execute. **Only this file has to be modified for this exercise.**
- `aux.c`, `aux.h`: these two files contain auxiliary routines and **must not be modified.**


The code can be compiled with the `make` command: just type `make` inside the `linked_list` directory; this will generate a `main` program that can be run like this:

```
$ ./main
```

3 Assignment

-  At the beginning, the `parallel_for_sweep` and `parallel_task_sweep` are a copy of `sequential_sweep`. Modify these routines in order to parallelize them as follows:
 - `parallel_for_sweep`: this parallel version must be based on the OpenMP `for` construct must not make use of the OpenMP `task` one.
 - `parallel_task_sweep`: this parallel version must be based on the OpenMP `task` construct

Make sure that the result computed by the three routines (sequential and parallel ones) is consistently (that is, at every execution of the parallel code) the same.

-  Report the execution times for the two implemented parallel versions with 1, 2 and 4 threads and compare them. Which version is faster? analyze and comment on your results. Report your answer in the `responses.txt` file.

Advice

- When implementing the parallel version based on OpenMP `for` remember that this construct only works if the number of iterations in the loop is known and therefore the provided code has to be modified in order to achieve the parallelization. Think about allocating new arrays that may help you achieve this.
- When using the OpenMP `task` construct, always think about data scoping to make sure input data has the correct value upon execution of the task and returned results do not go out of scope when the task is finished.