Livermore loops optimization with OpenMP

gcc -fopenmp ker21.c -m64 -lrt -lc -lm -o ker21

- ### Kernel 21:

As the code output shows for any “hello” word need to print “world” in follow.so we create a producer and consumer approach with max buffer size as “1”.in this way for any item that producer prints “hello” as output and waits for 1 second, consumer prints “world”.

We assign “MaxItems” as 5 so that producer create 5 items(print “hello”) for consumer.

- ### Commands:

```
gcc -IC:/MinGW/include/ -pg q1.c -lpthread -o q1.exe
```

```
q1.exe
```

- ### Comparison:

```
serial: 1000 microsecond
parallel: 2000 microsecond
```

in almost all cases serial method was faster (at least half time) or had equal time in compare, it seems that added overhead of parallel management and semaphore waits, make the process slower than serial manner.

Q2

- ### Approach:

The main PI calculation loop was divided based on number of threads and gave to threads for processing. An array of results created to store results of threads; each thread writes its own result in a cell of array.

In the end sum of all results used to calculate value of PI.

The information about each thread process gave to them as bellow structure:

```
double **sums = (double **)malloc(sizeof(double*) * threads_num);

typedef struct argument{
    int id;
    int size;
    int threads_num;
    double *sum;
}argument;
```

- ### Commands:

Compile command:

```
gcc -IC:/MinGW/include/ -pg pi.c -lpthread -o pi.exe
```

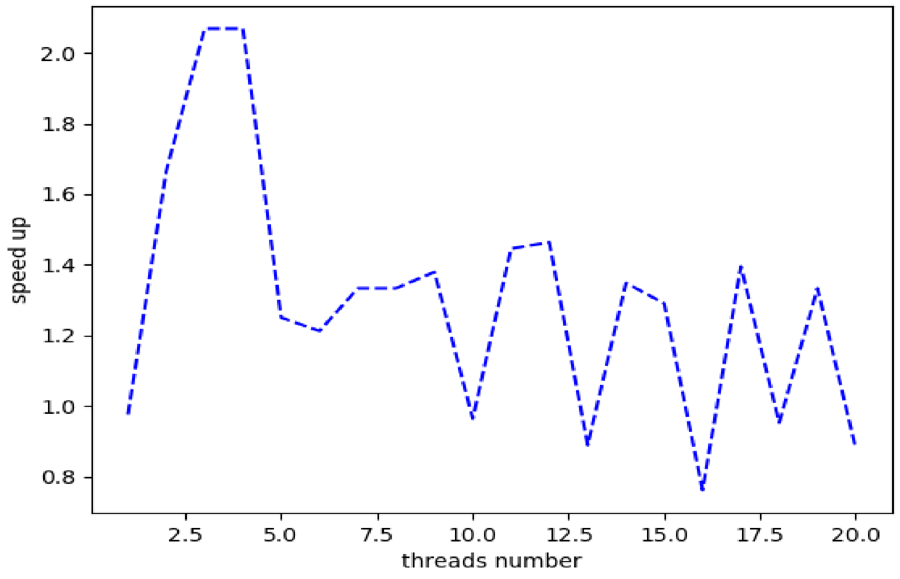
```
pi.exe <max threads num>
```

the input is max thread number. Code runs the process 40 time per each number of threads and return average PI, time and speed up value and store all speed ups into file “speedups.txt”.

```
plot.py
```

then by use of ***pyplot*** content of “speedups.txt” file used as input to plot speed up diagram.

- ### Comparison:



As the plot shows by increasing threads number, in small number of threads we have actual speed up and in the 2 or 3 number of threads is the best. It seems that from a point till end by adding more threads, the overhead of managing threads make the process slower than serial manner and doesn’t gave any benefits cause the main algorithm isn’t really complex and time consuming and 2 or 3 threads is enough.