Max Le
ID: 901223283
MTH 5050: Parallel Process
Assignment 4: Applications of OMP on Collatz Conjecture

1 Introduction

In this assignment, the goal is to parallelize a code that follows the Collantz conjecture. The conjecture states that starting with any integer n:

- if n is even, then n = n/2
- else if n is odd, then n = 3n + 1

This sequence then continues and according to the conjecture, all cycle will eventually terminates at 1.
For the current assignment, we are going to perform this procedure on a range of numbers, then for each of the number, we are going to try to record the highest n ever reached per range of numbers.
Our given range of numbers to be used are from 1 up to 2,000, 20,000, 200,000 and 2,000,000. The problems are to be completed using OMP and different parallel schedules are experimented.

2 Approach

For this problem, the algorithm for Collantz is relatively simple. Below is the snippet of the subroutine for this conjecture:

```
SUBROUTINE hotpo(ndum)
    IMPLICIT NONE
    INTEGER*8, INTENT(INOUT) :: ndum

    IF (mod(ndum,2) == 0) THEN
        ndum = ndum / 2
    ELSE
        ndum = (3 * ndum ) + 1
    END IF

    IF (ndum == 1) THEN
        ndum=1
    END IF
END SUBROUTINE hotpo
```

Then, the challenge is to decide which OMP schedulers is appropriate, as well as which variables to be made private and share. In this code, the loop variables and local max number are chosen to be private, while the start and end loop variables are shared. The tricky part would be to determine the global max,this is done via the reduction clause so that we can find the maximum over all threads. In code, it looks something like this:

```
!$OMP PARALLEL DO SCHEDULE(GUIDED,4) PRIVATE(iterNMAX,j,n,highN)&
!$OMP SHARED(maxHOTPO,nstart,nend) REDUCTION(max:GLOBAL_MAX)
DO iterNMAX = nstart,nend
    n = iterNMAX
    highN = 0
    DO j = 1,maxHOTPO
        !! local max
        IF(n > highN) THEN
            highN = n
        END IF

        !! calling hotpo subroutine
        call hotpo(n)

        IF (n == 1) THEN
            EXIT
        END IF
        !! global max
        IF(highN > GLOBAL_MAX) THEN
            GLOBAL_MAX = highN
        END IF
    END DO
END DO
!$OMP END PARALLEL DO
```

In terms of the schedulers, different types of schedulers will be investigated. These include: static, dynamics and guided. Additionally, chunk sizes are also investigated. For simplicity, chunk sizes of 1,2,3 and 4 are experimented for a specific number of thread: 4. This was chosen because 4 threads give steady performance across different problem sizes and scedulers.

3 Results

3.1 Maximum N

Firstly, below are the problem size the corresponding maximum N:

N	MAX N
2,000	1,276,936
20,000	27,114,424
200,000	17,202,377,752
2,000,000	156,914,378,224

3.2 Default Schedulers

Below are the results when using default schedulers:

N= 2,000

CORE	SPEED_UP_STATIC	SPEEDUP_DYNAMIC	SPEEDUP_GUIDED	SPEEDUP_THEORETICAL
1	1	1	1	1
2	2.04892018346509	1.71067329665139	1.35333056204578	2
4	2.72535383986511	3.01173238035838	2.5030992995424	4
8	2.86938768922888	3.85053879310345	3.61022354443739	8
16	0.527919781539104	3.85053879310345	0.370978224395059	16

N=20,000

CORE	SPEED_UP_STATIC	SPEEDUP_DYNAMIC	SPEEDUP_GUIDED	SPEEDUP_THEORETICAL
1	1	1	1	1
2	1.84136872462548	1.93452968040139	2.16823432589068	2
4	3.82480865911718	4.52192609416023	5.46314861112951	4
8	5.59746470969583	8.0071915463402	9.13488017367144	8
16	4.27293408072679	4.59155119252926	5.60212949569547	16

N=200,000

CORE	SPEED_UP_STATIC	SPEEDUP_DYNAMIC	SPEEDUP_GUIDED	SPEEDUP_THEORETICAL
1	1	1	1	1
2	1.62087521054844	1.70215521099083	1.73337941525506	2
4	3.08261233217805	3.39046854227403	3.62532073368413	4
8	3.88361730734444	6.00771894237201	6.63699166316962	8
16	6.36572986795432	7.80554751338753	7.83874583110833	16

N=2,000,000

CORE	SPEED_UP_STATIC	SPEEDUP_DYNAMIC	SPEEDUP_GUIDED	SPEEDUP_THEORETICAL
1	1	1	1	1
2	1.90572754032329	1.84725815704566	1.9256675415	2
4	3.50160223979762	3.63664572033789	3.80279410800741	4
8	5.44934806941363	6.19225101660861	6.09631824669698	8
16	8.19131461244874	8.74617559815515	8.46844395419051	16

In order to make it easier to visualize, the plots for the above data are produced per problem size as follow:

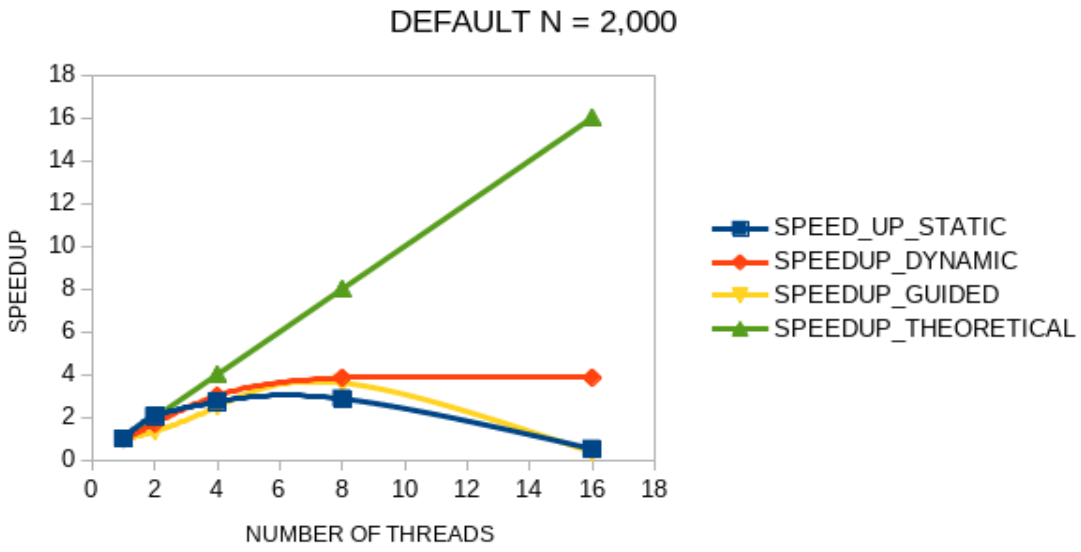

Figure 1: Default schedulers for N = 2,000

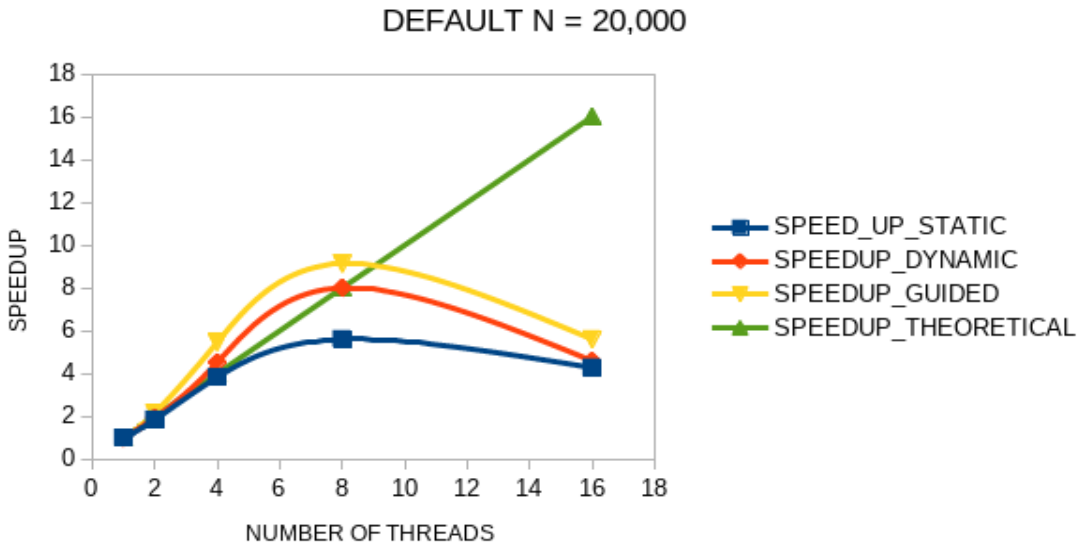

Figure 2: Default schedulers for N = 20,000

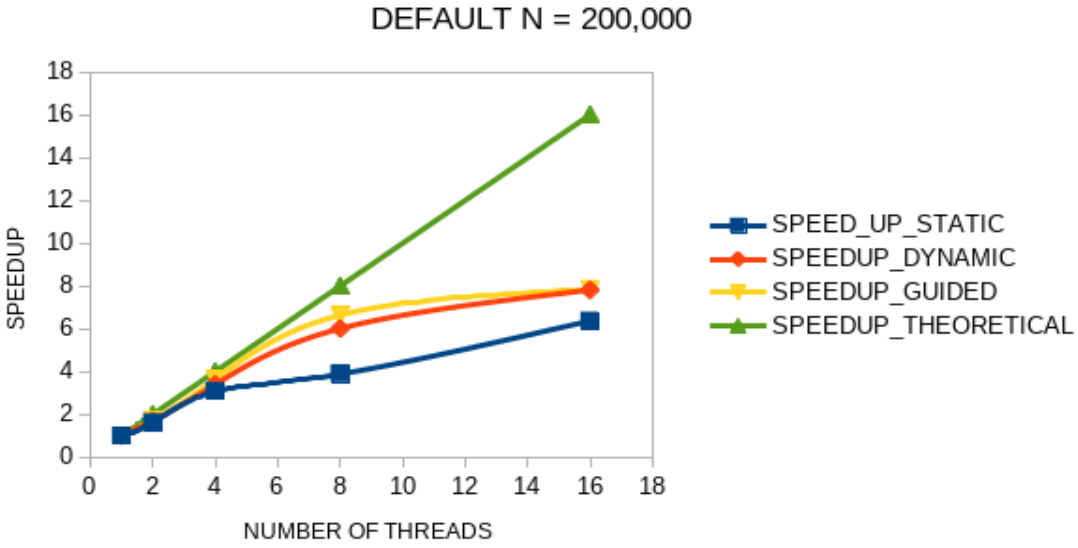

Figure 3: Default schedulers for N = 200,000

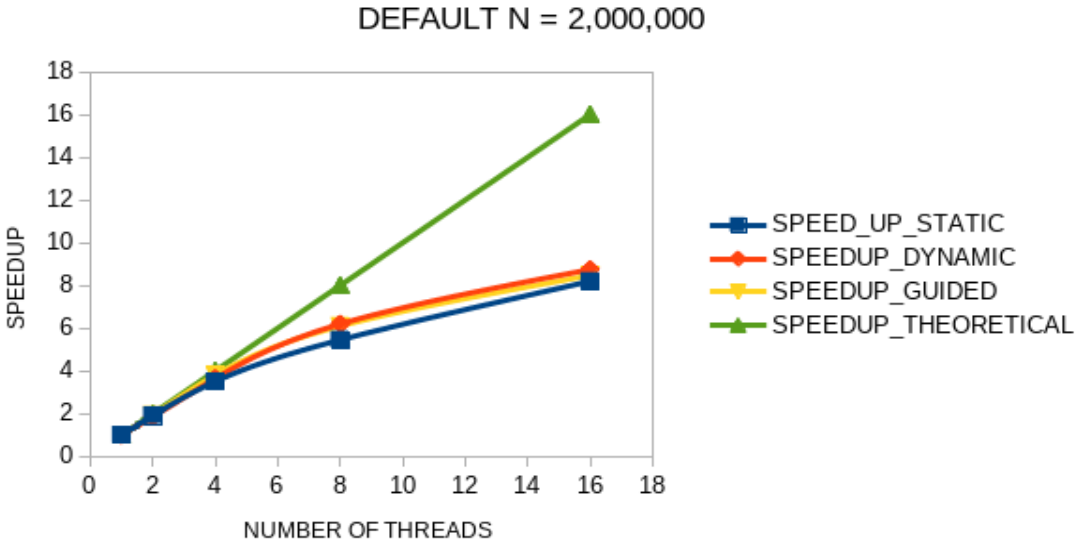

Figure 4: Default schedulers for N = 2,000,000

3.3 Chunk Sizes

N=2,000

CHUNK	SPEED_STATIC	SPEED_DYNAMIC	SPEED_GUIDED
1	6.89E-004	6.38E-004	6.31E-004
2	5.48E-004	5.88E-004	6.44E-004
3	6.34E-004	6.08E-004	6.23E-004
4	6.61E-004	5.64E-004	6.25E-004

N=20,000

CHUNK	SPEED_STATIC	SPEED_DYNAMIC	SPEED_GUIDED
1	7.72E-003	6.00E-003	5.46E-003
2	5.28E-003	4.65E-003	5.31E-003
3	5.54E-003	5.78E-003	5.30E-003
4	7.39E-003	4.73E-003	5.45E-003

N=200,000

CHUNK	SPEED_STATIC	SPEED_DYNAMIC	SPEED_GUIDED
1	7.88E-002	6.59E-002	7.05E-002
2	7.82E-002	5.69E-002	5.82E-002
3	7.05E-002	6.35E-002	7.26E-002
4	7.17E-002	5.82E-002	7.27E-002

N=2,000,000

CHUNK	SPEED_STATIC	SPEED_DYNAMIC	SPEED_GUIDED
1	6.26E-001	6.51E-001	7.01E-001
2	7.07E-001	6.30E-001	6.97E-001
3	6.44E-001	6.10E-001	7.01E-001
4	6.72E-001	6.24E-001	6.98E-001

Below are plots of chunk size vs. different speed from different schedulers.

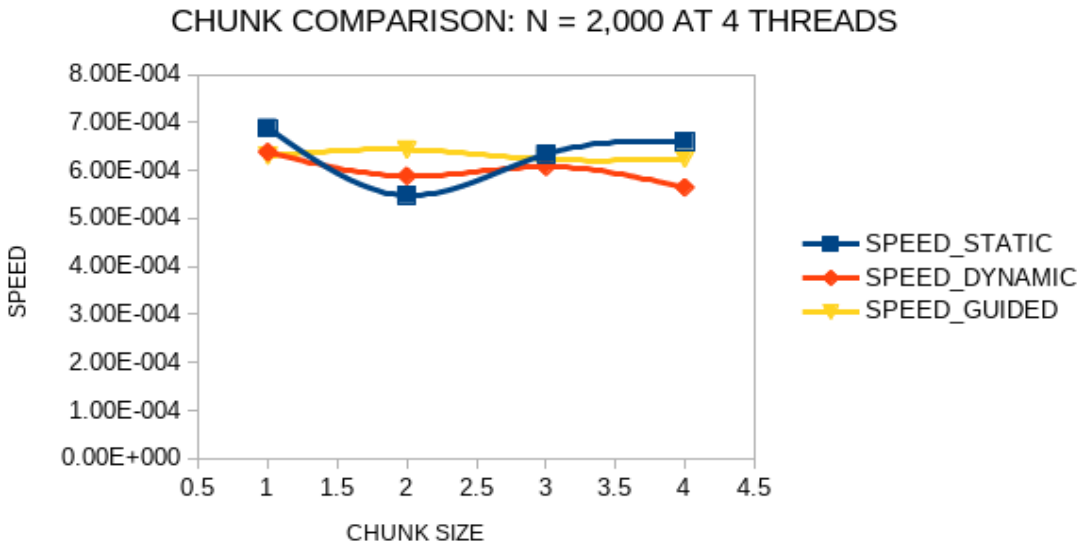

Figure 5: Chunk size comparison for N = 2,000

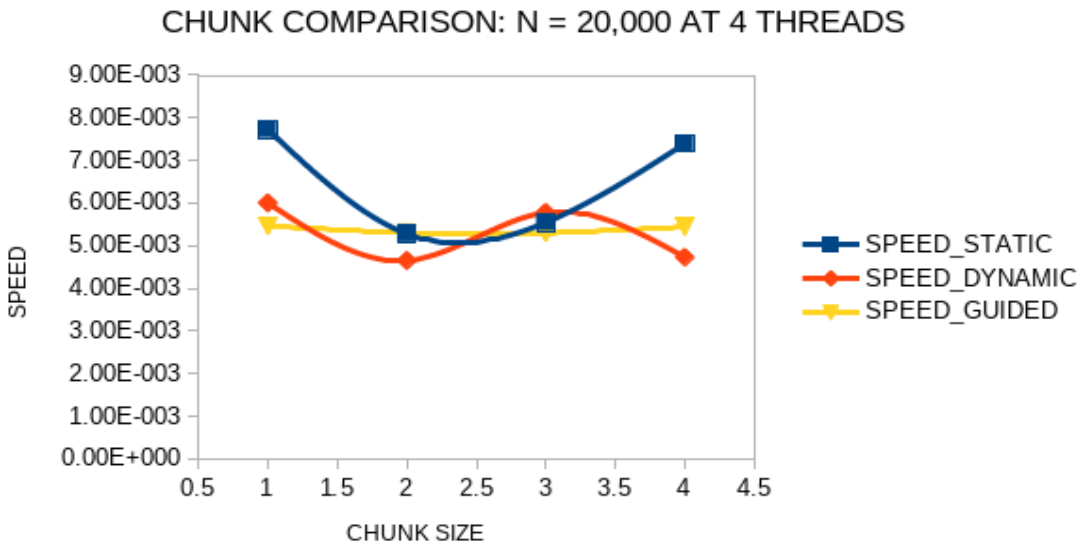

Figure 6: Chunk size comparison for N = 20,000

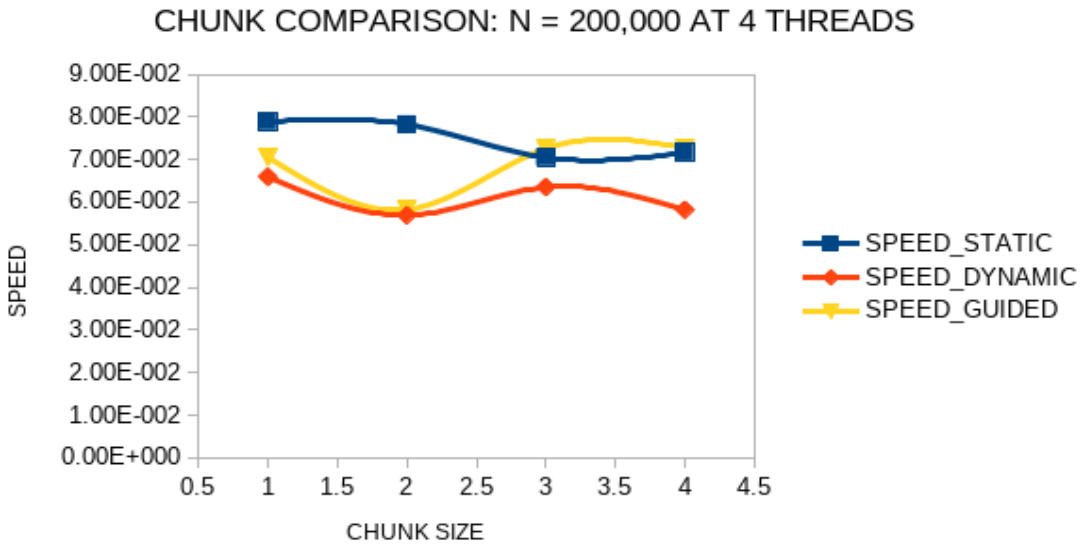

Figure 7: Chunk size comparison for N = 200,000

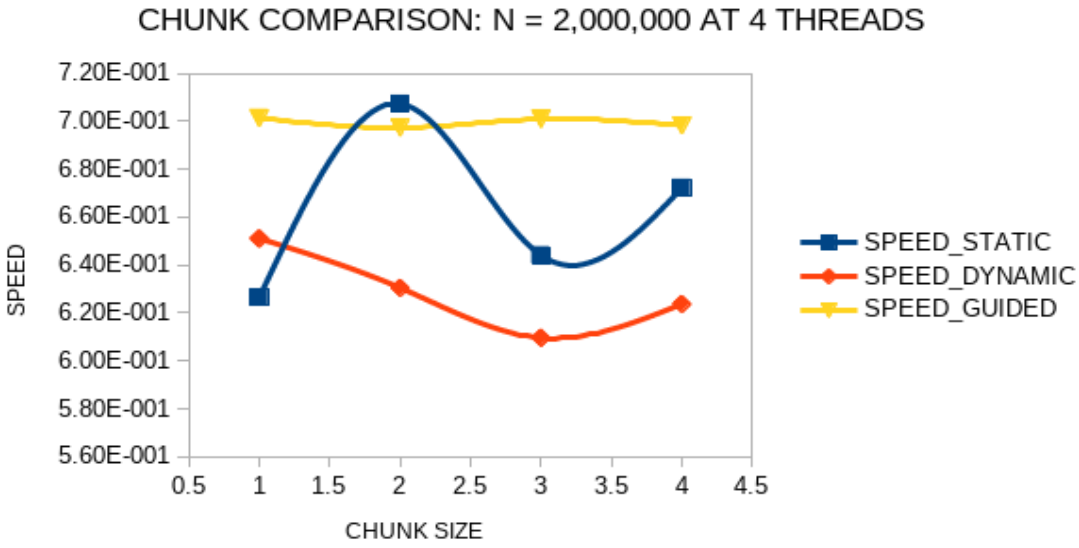

Figure 8: Chunk size comparison for N = 2,000,000

3.4 More Chunk Sizes

In order to investigate better, I tried to increase the chunk size for the extreme case: 2,000,000. The results are as follow:

N = 2,000,000

CHUNK	SPEED_STATIC	SPEED_DYNAMIC	SPEED_GUIDED
1	6.26E-001	6.51E-001	7.01E-001
2	7.07E-001	6.30E-001	6.97E-001
3	6.44E-001	6.10E-001	7.01E-001
4	6.72E-001	6.24E-001	6.98E-001
100	7.05E-001	6.99E-001	7.03E-001
2000000	2.65E+000	2.73E+000	2.67E+000

Clearly, increasing the chunk size does not really help. In fact, at the extreme when chunk size = problem size, then we have large overheads.

4 Discussions

Firstly, for default schedulers, at higher NMAX, dynamic and guided schedulers perform better than static scheduler. In fact, from Figures 1 to 4, we can see that the static scheduler is always the one that does not perform well compare to dynamics and guided. The reason for this is that the static scheduler does not perform well if individial iterations have widely different run times, which is the case for the Collantz conjecture.

Additionally, for a particular problem size, all default schedulers achieve speed up to to around 8 cores. At 16 cores, we ran into Blueshark's limitation of 12 cores per node. We can see this very clearly when N = 20,000; the recorded speedups at 16 cores are significantly lower than our results at 8 cores.
Secondly, the max speedup that I can get using Blueshark is using NMAX = 20,000 and using the default guided schedule with 8 cores. With this configuration, I got a speedup of around 9.13, according to figures and result tables in Section 3.2. This is actually better than theoretical speedup for 8 cores. In contrast, 16 cores do not give very good results and the reason might be that Blueshark is limited by 12 cores per node.

Finally, the chunk size also plays a role in this assignment. Surprisingly, higher chunk size tends to benefit the static scheduler. We can see this in Figure 5 to 8. Unfortunately, we will not get any speedup if we keep increase the chunk size. Section 3.4 shows this for NMAX = 2,000,000; at the extreme when chunk size = problem size, we have large overheads. At chunk size of 100, we did not see any improvements from the smaller chunk sizes.

5 Code

```
PROGRAM test

use omp_lib
IMPLICIT NONE
INTEGER*8 :: n,iterNMAX,j,highN, GLOBAL_MAX
INTEGER*8 :: maxHOTPO

INTEGER*8 :: nstart, nend

INTEGER :: num_threads
DOUBLE PRECISION :: tstart, tend, t_elapse

GLOBAL_MAX = 0
maxHOTPO = 1e8

num_threads = 4
call OMP_SET_NUM_THREADS(num_threads)

nstart = 1
nend = 200000

tstart = OMP_GET_WTIME()

!$OMP PARALLEL DO SCHEDULE(GUIDED,4) PRIVATE(iterNMAX,j,n,highN)&
!$OMP SHARED(maxHOTPO,nstart,nend) REDUCTION(max:GLOBAL_MAX)
DO iterNMAX = nstart,nend
    n = iterNMAX
    highN = 0
    DO j = 1,maxHOTPO
        !! local max
        IF(n > highN) THEN
            highN = n
        END IF


        call hotpo(n)

        IF (n == 1) THEN
            EXIT
        END IF

        !! global max
```

```
            IF(highN > GLOBAL_MAX) THEN
                GLOBAL_MAX = highN
            END IF

        END DO

    END DO
    !$OMP END PARALLEL DO


    tend = OMP_GET_WTIME()

    t_elapse = tend - tstart


    WRITE(*,*) ""
    WRITE(*,*) "PROBLEM SIZE: ", nend
    WRITE(*,*) "RUNNING WITH ", num_threads, " CORES"
    WRITE(*,*) "Time taken :",t_elapse
    WRITE(*,*) "GLOBAL MAX = ", GLOBAL_MAX



END PROGRAM test



SUBROUTINE hotpo(ndum)
    IMPLICIT NONE
    INTEGER*8, INTENT(INOUT) :: ndum

    IF (mod(ndum,2) == 0) THEN
        ndum = ndum / 2
    ELSE
        ndum = (3 * ndum ) + 1
    END IF

    IF (ndum == 1) THEN
        ndum=1
    END IF

END SUBROUTINE hotpo
```