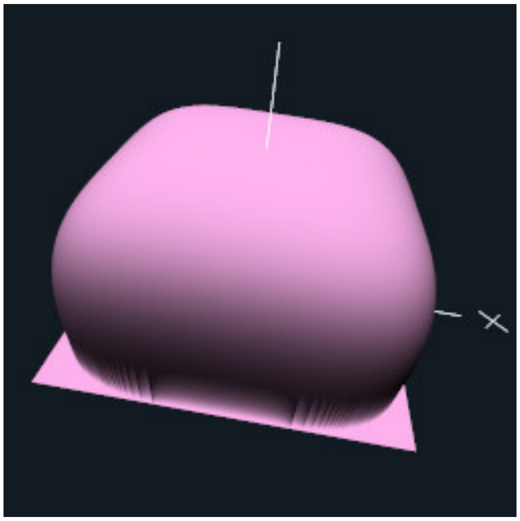
CS 475/575 -- Spring Quarter 2020

Project #2

Numeric Integration with OpenMP Reduction

100 Points

Due: April 26



Liang Zhao

933-667-879

zhaolia@oregonstate.edu

My program run on OSU Linux server: access.engr.oregonstate.edu. I used a bash script to automatically adjust the number of threads and Numnodes, and save the results in a csv file. Let us first look at the results of the experiment.

Number of Threads	Number of Numnodes	the total volume	Megatrials per Second
1	4	3.5334008	4.33
1	32	6.385921	3.74
1	128	6.4697914	3.57
1	500	6.4815373	3.66
1	800	6.48457	3.65
1	1000	6.4885488	3.67
1	1500	6.5042162	3.65
1	2000	6.5402517	3.59
1	2500	6.6064792	3.6
1	3000	6.6486855	3.59
1	4000	6.8581643	3.61
1	5000	5.3708572	3.65
1	6000	3.7295134	3.67
1	7000	2.7399201	3.66
2	4	3.5334003	5.62
2	32	6.3859096	5.44
2	128	6.4697566	7.09
2	500	6.4807563	7.12
2	800	6.4823575	7.06
2	1000	6.4837108	6.97
2	1500	6.4886594	7.13
2	2000	6.5026803	7.16
2	2500	6.5080214	7.17
2	3000	6.5487642	7.18
2	4000	6.6341262	7.15
2	5000	6.8130565	7.17
2	6000	6.9068213	7.18
2	7000	5.4798403	7.24
4	4	3.5334003	6.99
4	32	6.38591	14.23
4	128	6.4697676	13.95
4	500	6.4805365	13.75
4	800	6.481564	13.93
4	1000	6.4825373	14.02

4	1500	6.4850025	14.06
4	2000	6.4913969	14.05
4	2500	6.4958482	14.04
4	3000	6.5083532	14.04
4	4000	6.5108371	14.08
4	5000	6.5685358	14.1
4	6000	6.6580243	14.09
4	7000	6.7629137	14.1
8	4	3.5334034	5.15
8	32	6.3859096	22.62
8	128	6.4697742	21.16
8	500	6.4805102	22.43
8	800	6.4814205	27.58
8	1000	6.4819446	27.73
8	1500	6.4827099	27.46
8	2000	6.4843845	24.57
8	2500	6.4915013	26.95
8	3000	6.4916077	28.11
8	4000	6.4935131	27.62
8	5000	6.5341578	27.06
8	6000	6.5669332	26.06
8	7000	6.5975161	26.81
16	4	3.5334034	3.69
16	32	6.3859015	37.13
16	128	6.4697661	40.44
16	500	6.4804797	38.16
16	800	6.4813123	38.63
16	1000	6.4816022	20.68
16	1500	6.4820313	22.51
16	2000	6.4832878	24.17
16	2500	6.4832177	22.9
16	3000	6.4843888	24.55
16	4000	6.488771	22.7
16	5000	6.4998503	28.6
16	6000	6.5075426	35.5
16	7000	6.5254202	39.89
32	4	3.5334034	0.39
32	32	6.3859115	10.16
32	128	6.4697666	16.15
32	500	6.4804807	25.65

32	800	6.4811902	26.13
32	1000	6.481492	23.68
32	1500	6.4818153	24.48
32	2000	6.4821959	26.77
32	2500	6.4821424	29.12
32	3000	6.4828248	34.73
32	4000	6.4853129	33.15
32	5000	6.4883423	45.65
32	6000	6.4909062	45.23
32	7000	6.4972968	47.08
64	4	3.5334034	0.13
64	32	6.3859158	7.65
64	128	6.4697795	17.95
64	500	6.480494	36.06
64	800	6.4811916	43.55
64	1000	6.4814663	42.68
64	1500	6.4817314	30.42
64	2000	6.4819126	30.71
64	2500	6.482091	39.13
64	3000	6.4822369	45.66
64	4000	6.4827685	48.85
64	5000	6.4840722	51.54
64	6000	6.4851837	52.75
64	7000	6.4878588	54.56

1. Tell what machine you ran this on?

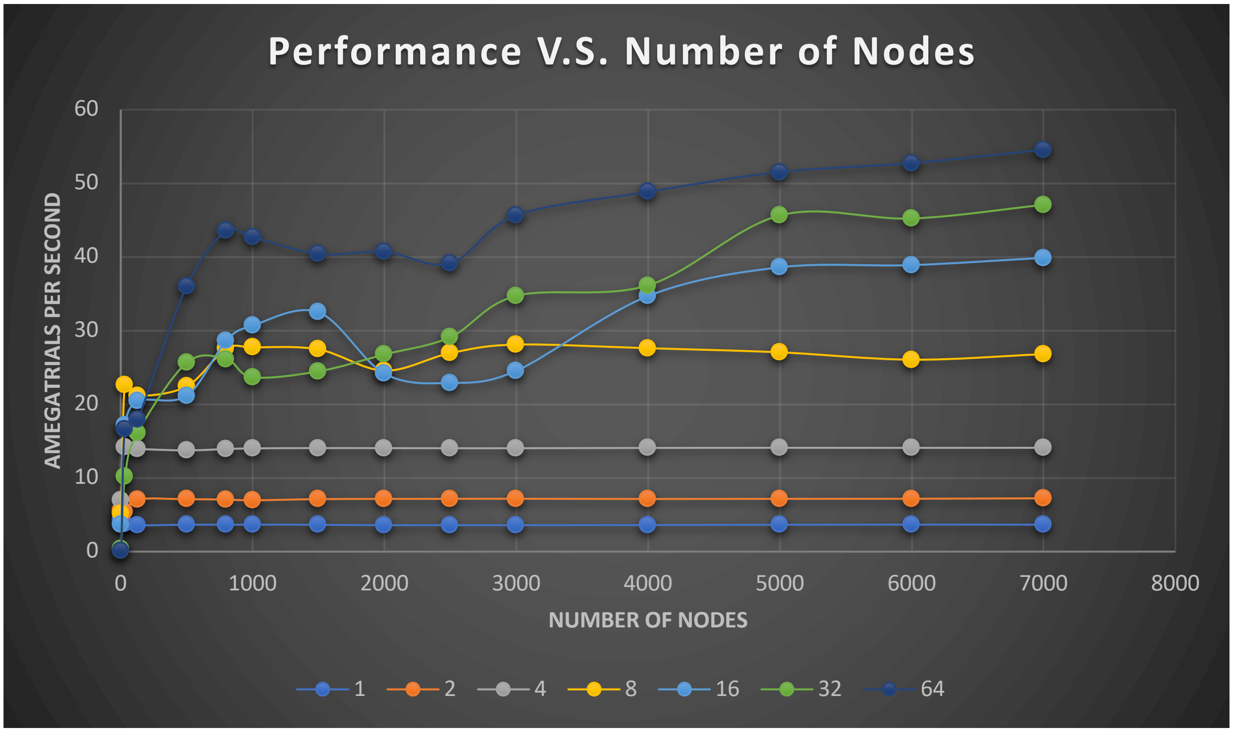
   My program run on OSU Linux server: access.engr.oregonstate.edu.

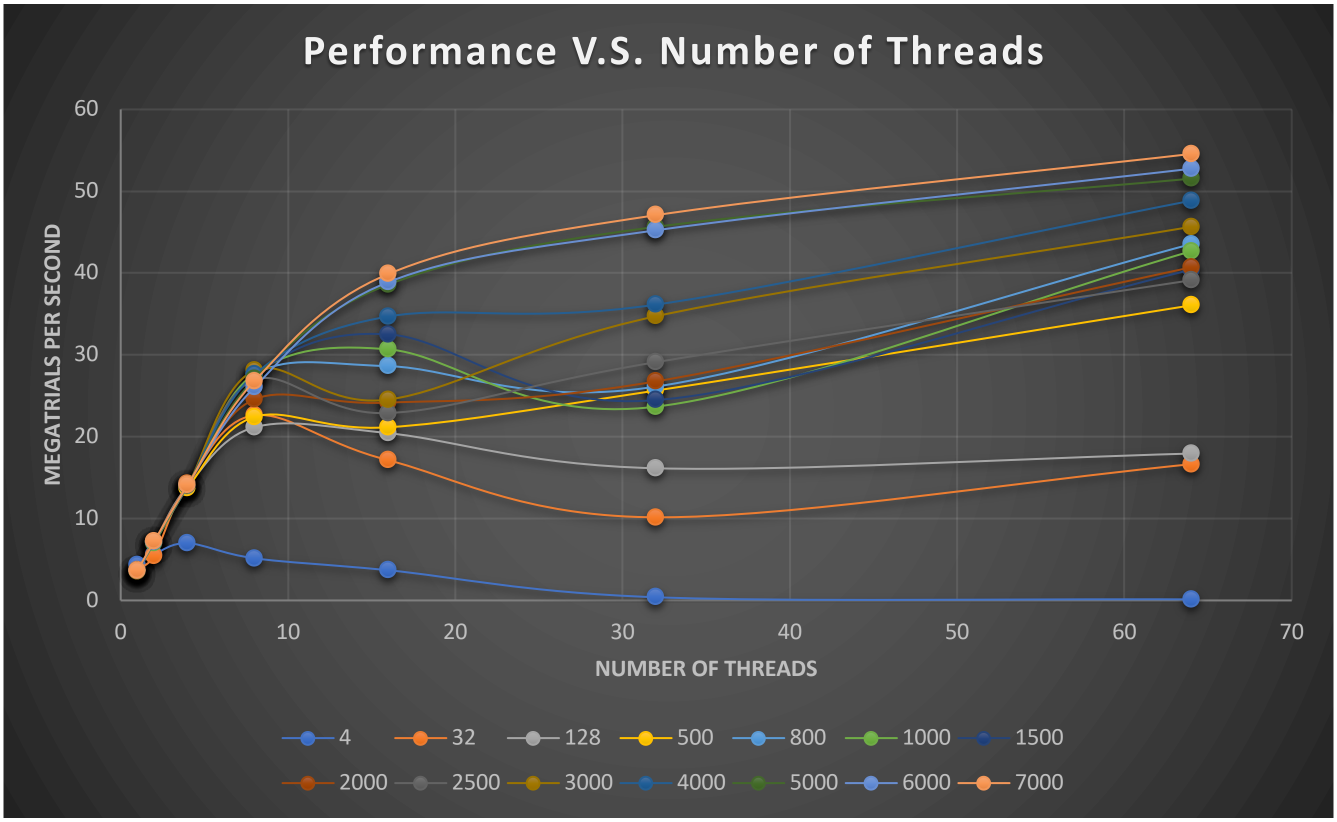
2. What do you think the actual volume is?

   I think the actual volume between the two surfaces is 6.49.

3. Show the performances you achieved in tables and graphs as a function of NUMNODES and NUMT.

Megatrials per Second		Number of Threads						
		1	2	4	8	16	32	64
Number of Nodes	4	4.33	5.62	6.99	5.15	3.69	0.39	0.13
	32	3.74	5.44	14.23	22.62	17.13	10.16	16.65
	128	3.57	7.09	13.95	21.16	20.44	16.15	17.95
	500	3.66	7.12	13.75	22.43	21.16	25.65	36.06
	800	3.65	7.06	13.93	27.58	28.63	26.13	43.55
	1000	3.67	6.97	14.02	27.73	30.68	23.68	42.68
	1500	3.65	7.13	14.06	27.46	32.51	24.48	40.42
	2000	3.59	7.16	14.05	24.57	24.17	26.77	40.71
	2500	3.6	7.17	14.04	26.95	22.9	29.12	39.13
	3000	3.59	7.18	14.04	28.11	24.55	34.73	45.66
	4000	3.61	7.15	14.08	27.62	34.7	36.15	48.85
	5000	3.65	7.17	14.1	27.06	38.6	45.65	51.54
	6000	3.67	7.18	14.09	26.06	38.9	45.23	52.75
	7000	3.66	7.24	14.1	26.81	39.89	47.08	54.56





4. What patterns are you seeing in the speeds?

Performance increases as the number of threads increases, and the performance remain substantially unchanged as the number of nodes increases. But performance starts to fall with higher core counts in the lower numbers of nodes.

5. Why do you think it is behaving this way?

We can see performance drops when the numbers of nodes because the code is taking up a more significant portion of our program when we only have a limited quantity of numbers to crunch. Performance is more potent with the same amount of threads, and a higher number of nodes because the multiple threads can appropriate more parallelizable code and higher parallel fraction.

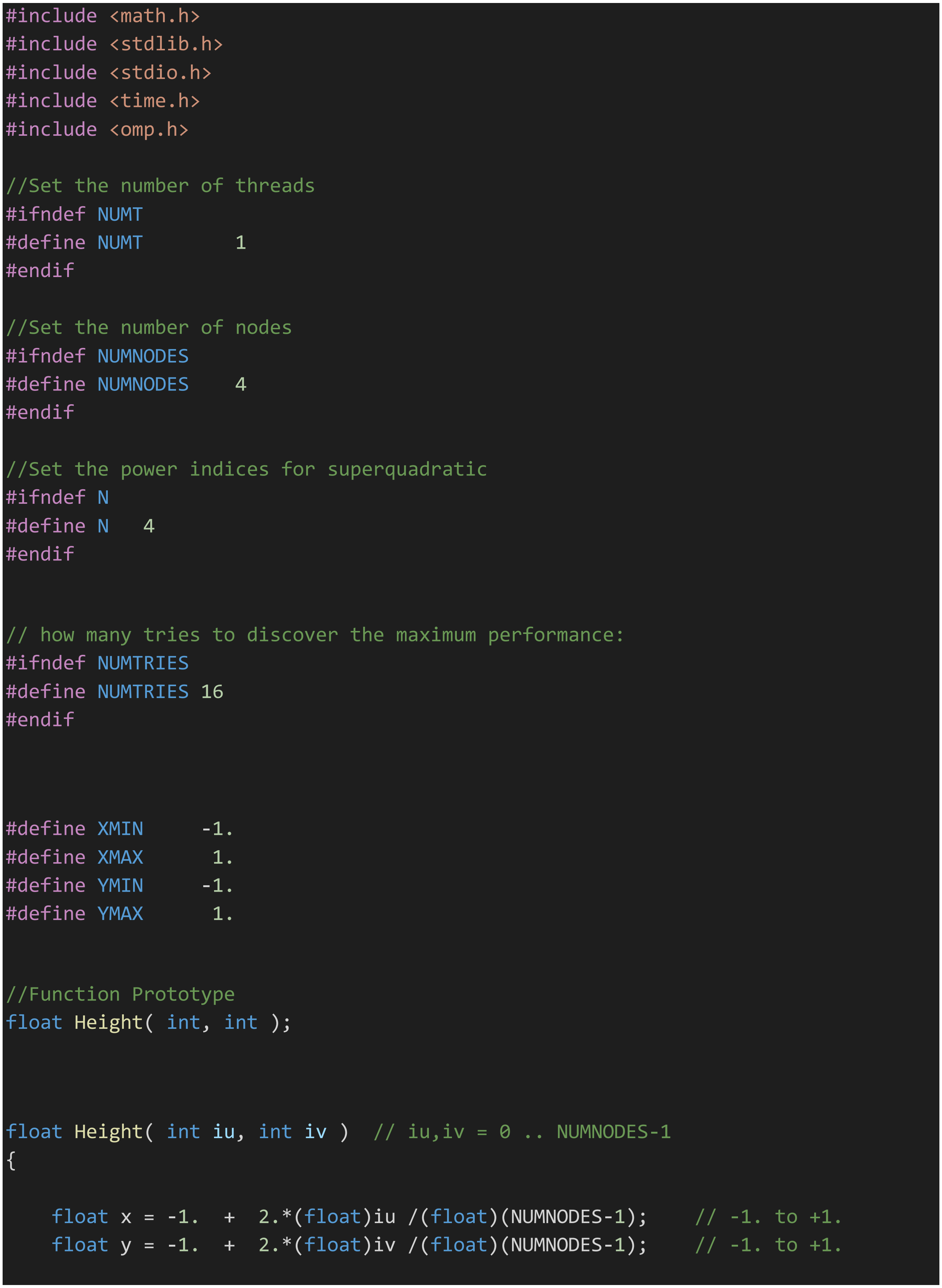
6. What is the Parallel Fraction for this application, using the Inverse Amdahl equation?

   I choose the commonly used 4 threads and 1 thread for speed comparison base on 5000 Nodes.

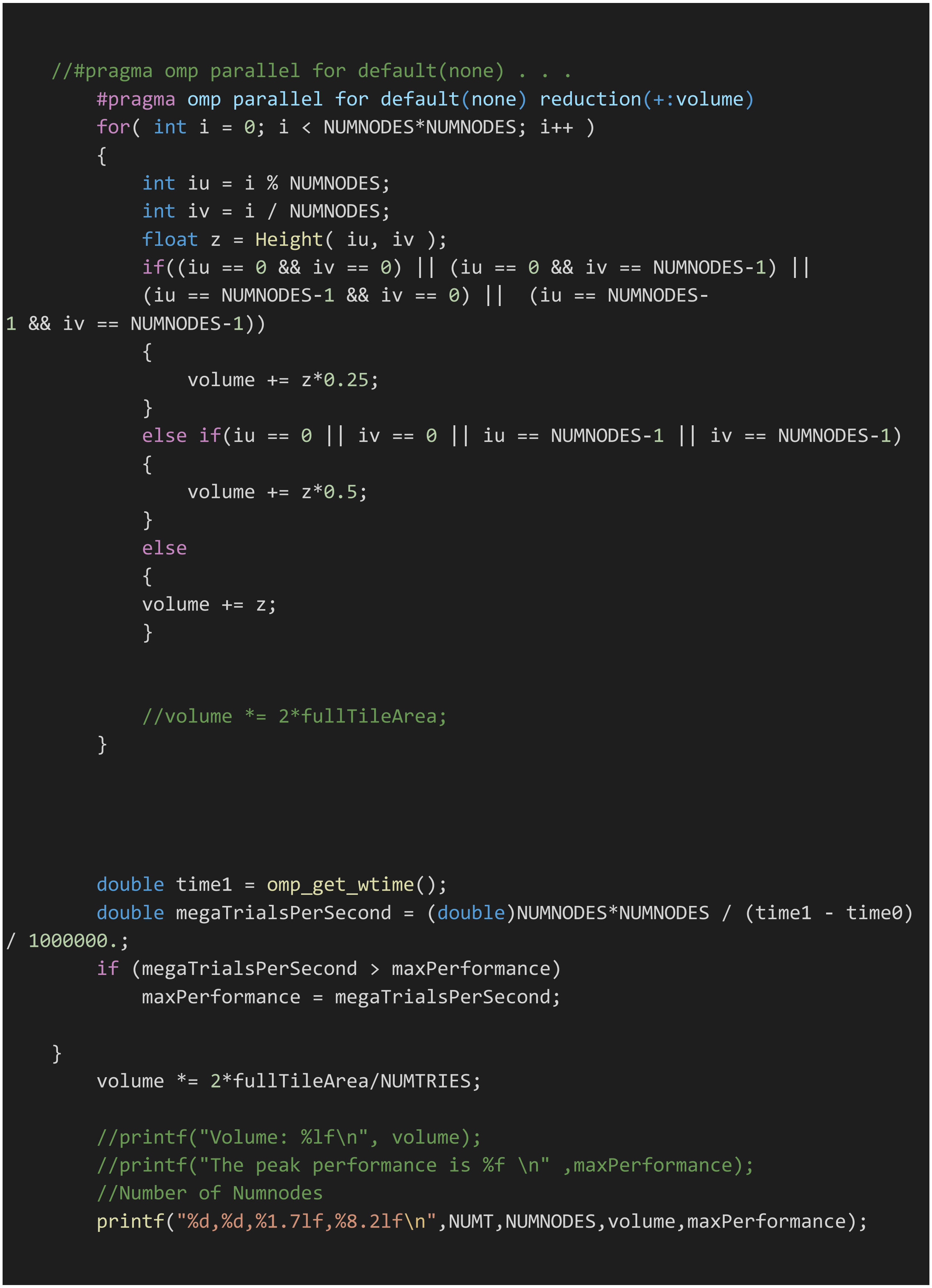
$$F_{Parallel=}\frac{\#\,Threads}{\#\,Threads-1}*\frac{T_{1\,Thread}-T_{n\,Threads}}{T_{1\,Thread}}=\frac{4}{3}*\frac{6.85-1.77}{6.85}=0.9888$$

7. Given that Parallel Fraction, what is the maximum speed-up you could ever get?

$$Max\ Speedup=\frac{1}{1-Fp}=\frac{1}{1-0.9888}=89.2857$$



```c
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <omp.h>

//Set the number of threads
#ifndef NUMT
#define NUMT        1
#endif

//Set the number of nodes
#ifndef NUMNODES
#define NUMNODES    4
#endif

//Set the power indices for superquadratic
#ifndef N
#define N   4
#endif


// how many tries to discover the maximum performance:
#ifndef NUMTRIES
#define NUMTRIES 16
#endif



#define XMIN     -1.
#define XMAX      1.
#define YMIN     -1.
#define YMAX      1.


//Function Prototype
float Height( int, int );



float Height( int iu, int iv )  // iu,iv = 0 .. NUMNODES-1
{

    float x = -1.  +  2.*(float)iu /(float)(NUMNODES-1);    // -1. to +1.
    float y = -1.  +  2.*(float)iv /(float)(NUMNODES-1);    // -1. to +1.
    float xn = pow( fabs(x), (double)N );
    float yn = pow( fabs(y), (double)N );
    float r = 1. - xn - yn;
    if( r < 0. )
            return 0.;
    float height = pow( 1. - xn - yn, 1./(float)N );
    return height;
}




int main( int argc, char **argv )
{


    //Verify OpenMP support
    #ifndef _OPENMP
        fprintf( stderr, "No OpenMP support!\n" );
        return 1;
    #endif


    //Set the number of threads for the for loop
    omp_set_num_threads(NUMT);

    float volume = 0.;

    // the area of a single full-sized tile:

    float fullTileArea = (  ( ( XMAX - XMIN )/(float)(NUMNODES-1) )  *
                ( ( YMAX - YMIN )/(float)(NUMNODES-1) )  );

    // sum up the weighted heights into the variable "volume"
    // using an OpenMP for loop and a reduction:

    float maxPerformance = 0.;

    for (int t = 0; t < NUMTRIES; t++)
    {

        double time0 = omp_get_wtime();

    // sum up the weighted heights into the variable "volume"
    // using an OpenMP for loop and a reduction:
```



```
    //#pragma omp parallel for default(none) . . .
        #pragma omp parallel for default(none) reduction(+:volume)
        for( int i = 0; i < NUMNODES*NUMNODES; i++ )
        {
            int iu = i % NUMNODES;
            int iv = i / NUMNODES;
            float z = Height( iu, iv );
            if((iu == 0 && iv == 0) || (iu == 0 && iv == NUMNODES-1) ||
            (iu == NUMNODES-1 && iv == 0) ||  (iu == NUMNODES-
1 && iv == NUMNODES-1))
            {
                volume += z*0.25;
            }
            else if(iu == 0 || iv == 0 || iu == NUMNODES-1 || iv == NUMNODES-1)
            {
                volume += z*0.5;
            }
            else
            {
            volume += z;
            }


            //volume *= 2*fullTileArea;
        }





        double time1 = omp_get_wtime();
        double megaTrialsPerSecond = (double)NUMNODES*NUMNODES / (time1 - time0)
/ 1000000.;
        if (megaTrialsPerSecond > maxPerformance)
            maxPerformance = megaTrialsPerSecond;

    }
        volume *= 2*fullTileArea/NUMTRIES;

        //printf("Volume: %lf\n", volume);
        //printf("The peak performance is %f \n" ,maxPerformance);
        //Number of Numnodes
        printf("%d,%d,%1.7lf,%8.2lf\n",NUMT,NUMNODES,volume,maxPerformance);
    return 0;


}
```