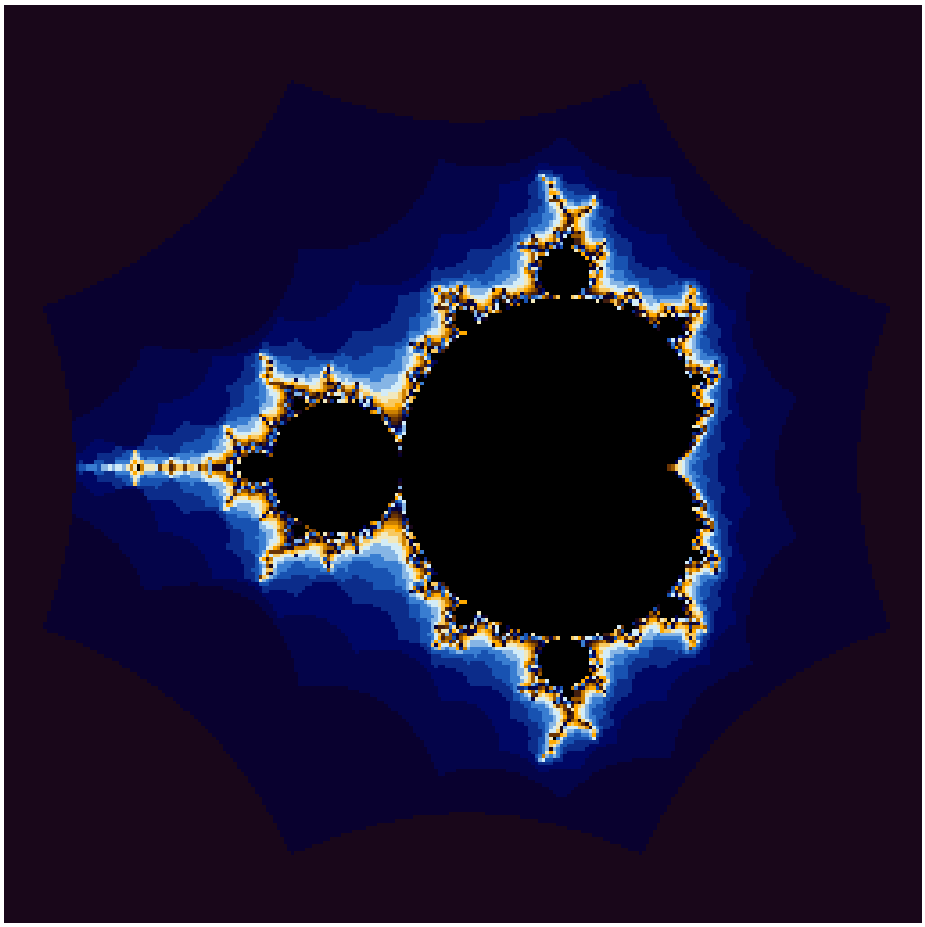
CS-301

Parallel generation of Mandelbrot fractal



Unnati Parekh 201501406
Krishna Manvar 201501412

1 Introduction

Fractals are objects with dimensions that cannot be expressed as whole numbers. They have a finite area bounded by an infinite curve and are characterized by their self symmetry. Fractals are generated by calculating certain formulas over and over again, feeding the results of one step back into the next step.

The Mandelbrot set, the most well-known fractal object first described by Benoit Mandelbrot, is created by iterating a simple formula over complex numbers. Mandelbrot set is defined as complex points c, where $|Z_\infty|$ is finite, with $Z_0 = 0$ and $Z_{n+1} = Z_n^2 + c$.

2 Implementation Details

2.1 Brief and clear description about the Serial implementation

With real computers we can't deal with infinity. It can be proven from the mathematical definition that if the absolute value of Z ever gets bigger than 2, it will never return to a place closer than 2, but it will actually rapidly escape to infinity. Hence, if $|Z_n| > 2$, point diverges.

Now as the boundary point changed to 2, we need to iterate Z_n only a few times to check if it is present in the set or not. Here, we have kept the maximum iteration value as 50. The maximum iteration value only changes the resolution accuracy of the image. If Z_n did not escape infinity after 50 iterations, it is considered to be part of the set. But many points may need more iterations to escape to infinity.

When we draw the Mandelbrot set in an image, we need to set the equivalence between pixel coordinates and complex numbers. Each pixel in our image has to represent a complex number. To start creating a fractal image, we choose an arbitrary point C from the complex plane and plug it into the following simple iteration formula:

$$Z_0 = 0$$
$$Z_{n+1} = Z_n^2 + C$$

Then we will color that pixel according to the number of iterations it takes for a point C, before Z_n diverges.

2.2 Brief and clear description about the implementation of the parallelisation approach

Fractal image generation is computationally expensive. For example, to generate a highly accurate fractal image for an image size of $1024*1024$, at a resolution of 1 dot per pixel, we have to apply the iteration formula to $1024*1024 = 1,048,576$ points, and perform up to 1,500,000 iterations on each of those points, leading to an upper bound of 1,572,864,000,000 computations.

All the computations for one point should be performed on the same processor to avoid the need for communication among processors. As all the points are individually checked, this is an embarrassingly parallel problem. If we divide the image matrix into blocks, some processors remain idle while others have a greater workload. Hence, we divide the workload amongst the processors in such a way that each is concerned with alternating rows in the image matrix. For this we parallelize the outer loop.

2.3 The CPU information on which these codes are run

- Model name: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
- Architecture: $x86-64$
- CPU(s): 12
- Thread(s) per core: 1
- Core(s) per socket: 6
- Socket(s): 2
- L1d cache: $32K$
- L1i cache: $32K$
- L2 cache: $256K$
- L3 cache: $15360K$

3 Complexity and Analysis

3.1 Complexity of serial code

Complexity of serial code is O(n*n*k).

Here n is the size of image and k is the number of maximum iterations.

3.2 Complexity of parallel code

Complexity of parallel code is O(n*n*k).

Here n is the size of image and k is the number of maximum iterations.

3.3 Theoretical Speedup

Theoretical speedup is p.

Here p is the number of processors.

3.4 Number of memory accesses

For both the paradigms, the number of memory accesses is n*n.

Here n is the image size. For each pixel in the image, we calculate if it is present in the set or not and colour the pixel according to the number of iteraions it takes to diverge.

3.5 Number of computations

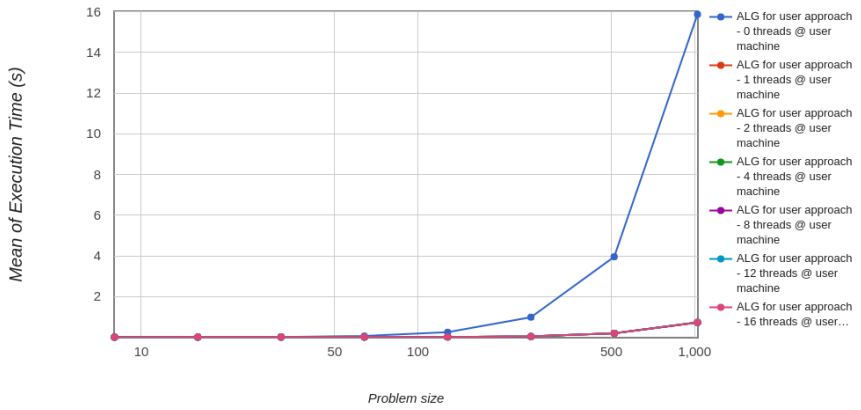
For serial implementation, number of computations is n*n*k.

For parallel implementation, number of computations is n*n*k/p

Here, n is size of image, k is maximum iterations and p is the number of processors.

4 Curve Based Analysis

4.1 Execution time related analysis



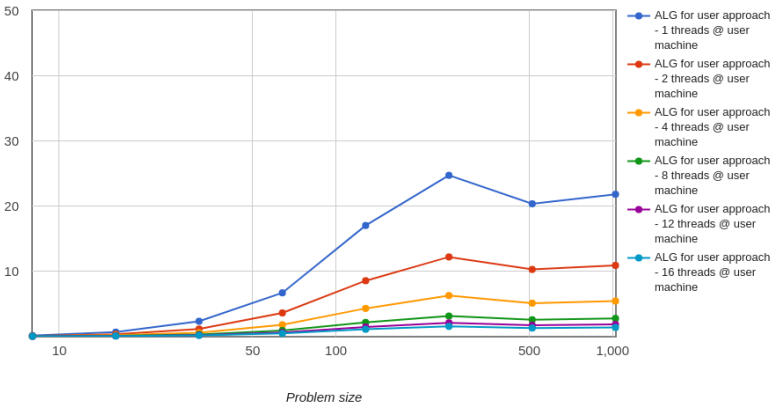
Execution time for parallel algorithm is very less than that of serial, as it is an embarrisingly parallel problem.

4.2 Karp flat Analysis

x	2	3	4	5	6	7	8	9	10	11	12	13
y	0.222	0.023	0.067	0.044	0.055	0.06	0.035	0.05	0.034	0.047	0.36	0.44

Karp-Flatt metric can be used to calculate an experimentally determined serial fraction of a parallel computation
Here, we have calculated the Karp-Flatt metric for problem size 1024*1024.
Here, we see the values almost remain the same for different threads so the code is efficient and there is very less serial fraction.

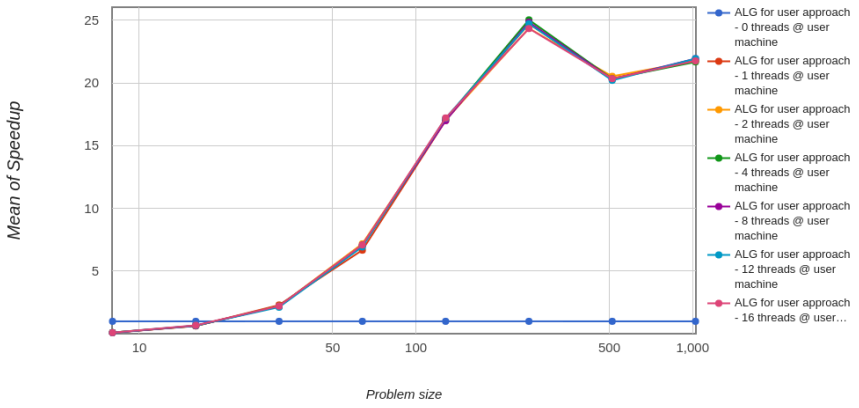
4.3 Efficiency related analysis



Efficiency decreases with increase in number of processors, as problem size increases.

The speedup remains almost the same, so as the number of processors increase, there is a decrease in efficiency.

4.4 Speedup related analysis



As we have mentioned above, this is an embarrassingly parallel problem. So we get a high speedup for this problem.

As problem size increases, speedup increases for a fixed number of threads, which is evident from the above graph. With increasing problem size, the overheads involved are less than the gain in performance due to parallelization.

However as number of threads increase, the overheads involved in creating, synchronization of threads increase.

For a fixed problem size, speedup usually increases as the number of threads increase. However if the problem size is small, then the overheads due to creating threads dominate the gain in performance due to parallelization.

5 Further Detailed Analysis

5.1 Major serial and parallel overheads

Points in the mandelbrot set, iterate a large number of times, while those outside the radius of 2, only need to iterate for 2-3 times, before we discard them.

Since no information from other points of the image goes into the iteration formula, there is no need for communication among the points.

5.2 Load balance analysis

- Mandelbrot set generation is a good example of a non balanced problem. Default work sharing break N iterations into $\sum N/nthreads$ contiguous chunks and assigns them to threads.
- But some points diverge in 2-3 iterations and the threads handling them would remain idle afterwards.
- Hence, instead of using the default scheduling construct, where each thread gets a big chunk of the data, we can use dynamic or static scheduling to assign more than one chunk to a single thread.
- schedule(static, m) gives m consecutive loop elements to each thread instead of a big chunk.
- With schedule(dynamic, m), each thread will work through m loop elements, then go to the OpenMP run-time system and ask for more.
- Load balancing is (possibly) better with dynamic, but larger overhead than with static.

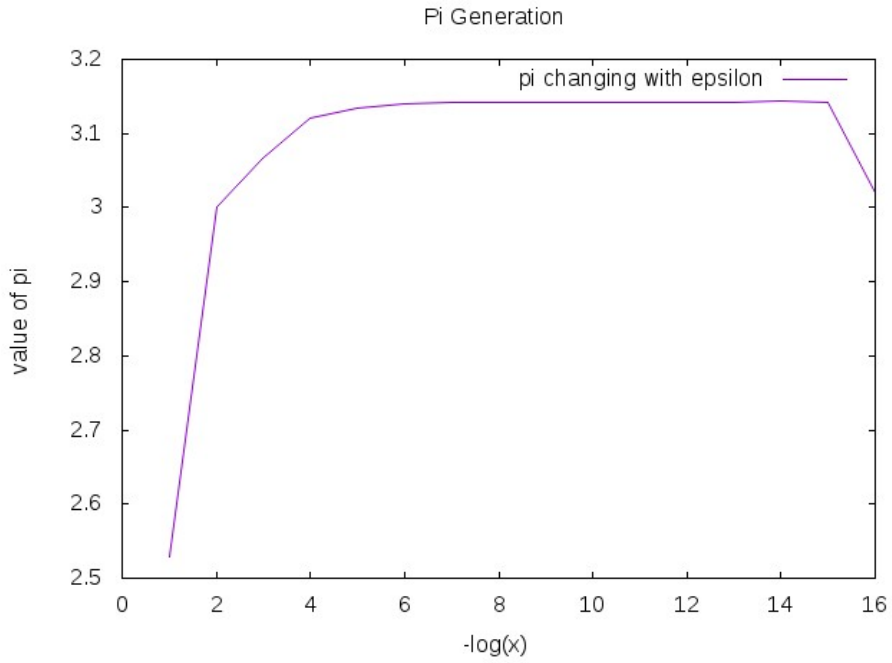
6 Applications

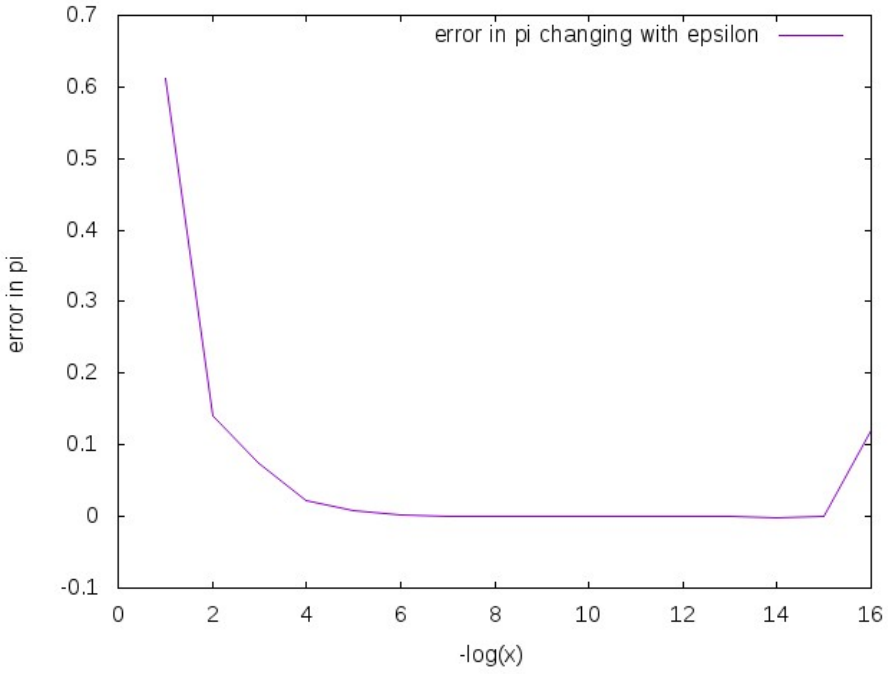
6.1 Estimating the value of π

For various points on the real line very close the Mandelbrot Set, if we calculate the number of iterations required for the function value to come close to converging, the number of iterations is evidently the various digits in π. Here, we take the points on the real line which are present very close to the cusp of the plot viz. 0.25.

Now for any complex number $0.25 + \epsilon$, where ϵ is a very small positive number, as we decrease the value of epsilon, the number of iterations it takes to diverge, increases. We obtain the value of π by the following formula, $n\sqrt{\epsilon}$.

Here, n is the number of iterations before C diverges.





6.2 Fractal image compression

Data compression techniques using fractals takes advantage of similarities within an image, hence using it's inherent self similarity property. Image compression is done based on the Collage Theorem which states that if the error difference between the target image and the transformation of that image is less than a certain value the transforms are an equivalent representation of the image.

The idea is to look for parts in a picture that resembles other parts in the same picture. The picture is divided into range blocks (small) and domain blocks (big). Use a domain block to describe a range block. For each range block, search for the domain block that most closely resembles the range block. Transformations such as scaling, translation, rotating, sharing, scaling etc and adjustment of brightness/contrast are used on the domain block in order to get the best match.

To decompress, we could use any image as starting image. Apply the transformations and translations the arbitrary image (usually just a grey background) iteratively until an image is produced that looks approximately like the original.

6.3 Fibonacci sequence

While noticing the different characteristics of the Mandelbrot plot, there is one striking observation that we can make. Starting from the two larger bulbs of the plot with two and three antennas each, we find the bulb in the middle of the two bulbs which will have number of antennas as the next Fibonacci number $(2 + 3 = 5)$. Similarly, the next Fibonacci number can be found by the number of antennas in the bulb in between the bulb with 3 antennas and 5 antennas, and the process can be continued similarly.

7 References

- $https://wiki.scinet.utoronto.ca/wiki/images/e/ef/SS_Openmp.pdf$
- $http://matthiasbook.de/papers/parallelfractals/introduction.html$
- $https://www.youtube.com/watch?v = d0vY0CKYhPY$