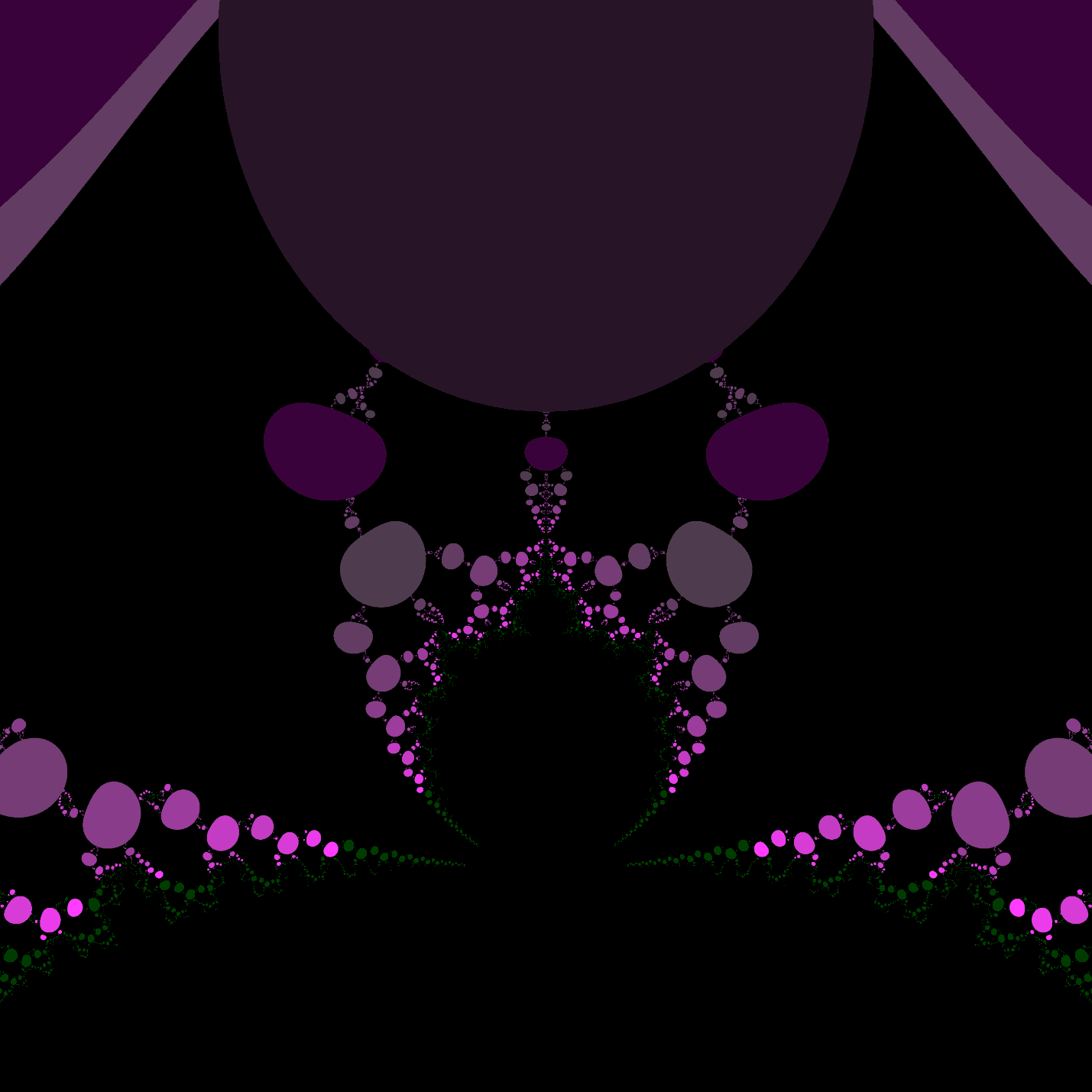
**Fractal Generator**

Like the Mandelbrot set which is used to generate images based on the value of a number and if it satisfies a condition or not, different fractals can be generated using different conditions set. This fractal generation program was tested for the pattern generated by repeatedly applying the formula:  
  
**z = tan(z2 )+ c**

where **c** is the complex “value” of the pixel’s coordinates after a transformation has been applied (the transformation is necessary to fit the fractal into the picture properly). **z** is calculated for every pixel in the picture in a loop. The absolute value of the complex number **z** is tested, and if it is greater than 2 at any iteration, we assume that the sequence will escape to infinity. Using the resulting **z** value, a color is assigned to the pixel. Additionally, after 34 iterations have passed, we stop calculating **z** and the pixel remains black as RGB(0,0,0) will be used for it.

To parallelize this program, the **z** value for each pixel can be calculated separately. No pixel’s color is dependent on a previous pixel’s, so the for-loop that goes over each pixel can easily be parallelized with an OpenMP directive without having to adjust the code for dependencies. Using OpenMP also ensures that there are minimal changes to the original sequential code; only one directive and some extra loops to time the output were required.

The 3000x3000 fractal generated is shown as follows:



Some pixels were colored green due to the conversion of PPM to PNG format (PPM images cannot be included in .docx documents). The original PPM file in an image editor such as GIMP is shown with better colors instead of the green shown above.

The time (in milliseconds) for total number of threads (1 to 24) in each iteration is shown in the following table:



The calculated speedup and efficiency graphs are as follows:

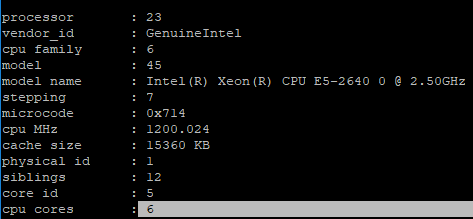
The speedup is almost consistently increasing, but the rate of increase slows down around number of threads = 14. This is expected due to communication costs between cores also increasing as number of employed threads increase. The speedup with 24 threads is over 10, which is a great improvement from the sequential program.

Efficiency is overall decreasing, which can also be attributed to core communication costs.

Overall, the speedup results show that a fractal generator of this kind definitely benefits from being parallelized.

**Specifications**

* Using ***/proc/cpuinfo***, we have the following output:



* **qmake** (version shown below) was used to compile the project.



* The speedup and efficiency graphs are plotted for thread values ranging from 1 to 24 for an image size of 3000x3000. OpenMP function *omp\_set\_num\_threads()* is usedto set the number of threads in the loop and *QTime* is used to time each iteration. For each thread number, three tries are done, and the average time of those three tries is printed out. This is to reduce the fluctuation in the graphs due to unusual results.

**Machine Configurations**

* Intel(R) Xeon(R) CPU E5-2600
* Clock Speed: 2.50 GHz
* RAM: 65916328 kB
* Number of processors: 24 (with hyperthreading)
* Number of CPU cores: 6

**Software Specifications**

* Operating System: Ubuntu 18.04.1 LTS
* Compiler: GNU C++ compiler