PARALLEL PROCESSING AND THE MANDELBROT SET

Carmen PUGHINEANU

"Ştefan cel Mare" University, Suceava, Romania

Abstract*: The theory of fractals and chaos is one of the newest branches of mathematics and informatics. The history of fractals with the publication of Benoit Mandelbrot's "A theory of the fractals series". He wanted to suggest a group that is much more "irregular" than those considered in classical geometry; the more increased it is, the more irregularities become visible. As a definition, the fractals are complex geometrical figures generated with the help of a mathematical pattern that has three characteristics: they are infinitely complex, have signs of self-similarity and contain areas of orders and of chaos. The Mandelbrot Set constitutes a special category of the fractals.*

Key words*: efficiency, fractal, iteration, parallel processing, parallel speed.*

1. THE MANDELBROT SET

To illustrate the distributed calculus the fractal that bears the name of Mandelbrot shall be used.

The fractals [1] are generated with certain recursive calculus formulae. For example, the best known fractal described by Benoît Mandelbrot, is iteratively created with a simple formula over the groups of complex numbers.

A complex number C is formed of two components – the real part ReC and the imaginary part ImC. The imaginary part is multiplied by the constant i, defined as $i^2 = -1$.

$$C = \mathrm{Re}C + \mathrm{Im}C \cdot i \qquad (01)$$

The modulus of a complex number is defined as being the distance from the point to the origin and it is calculated with the formula:

$$|C| = \sqrt{(\mathrm{Re}\,C)^2 + (\mathrm{Im}\,C)^2} = \sqrt{x^2 + y^2} \qquad (02)$$

An arbitrary point C in the complex plan is selected and used in the iterative formula:

$$Z_0 = 0$$

$$Z_{n+1} = Z_n^2 + C \qquad (03)$$

For the first iteration, it is replaced with Z_0, which is the origin, it is squared, the constant C chosen before is added and we obtain a new point Z_{n+1}. This point is again squared; C is added again and so on. This is repeated several times and the behavior is observed.

For example if we selected $C = -1 + i$, we reach a series of values for Z presented in Table 1:

Table 1

n	Re(Z_n)	Im(Z_n)	$\lvert Z_n \rvert$
0	0	0	0
1	-1421	18109	18164.7
2	4,39E+72	1,33E+73	1,40E+73
3	-1.#IND	-1.#IND	-1.#IND

One can notice that Z tends goes to infinity after a very small number of iterations.

In fact, the result is predictable, because we square and add numbers. Yet, if we try another choice for C, for example $C = 0.25 - 0.5i$, we obtain the results from Table 2:

Table 2

N	Re(Z_n)	Im(Z_n)	$\lvert Z_n \rvert$
0	0	0	0
1	0.472656	-0.6875	0.834302
2	0.113859	-0.376507	0.393346
…	…	……	…
20	0.440247	-0.66811	0.800118
21	0.129705	-0.364427	0.386821
…	…	…	…
998	0.172002	-0.409976	0.444595
999	0.602197	-0.72362	0.941417
1000	0.219192	-0.442107	0.493461

This time Z remains in an interval close to the origin, when the same iteration formula is applied. If the iteration is executed several times one will notice that Z converges to a certain point. Thus, Z behaves differently depending on the initial selection of C.

Further on, we shall apply the iteration formula for any point C that is a complex number and will color it black if Z converges and in white if Z goes to infinity.

The result is represented in the Figure 1 and is in fact the multitude of points from the Mandelbrot Set.

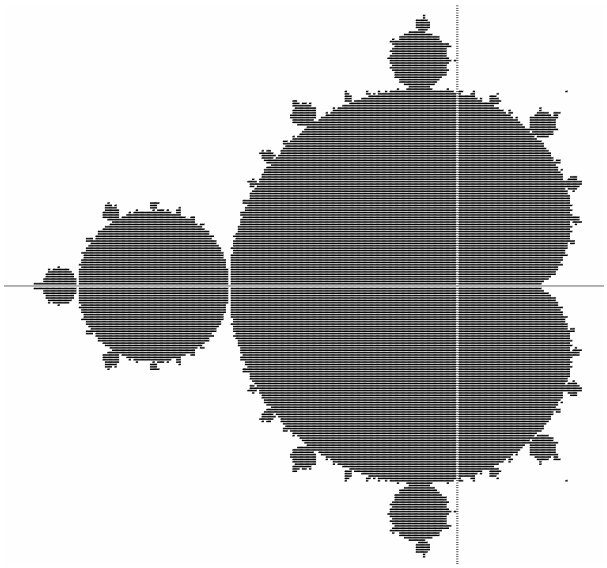

Fig. 1 The Mandelbrot Set

2. THE PARALLEL CALCULUS

Generating the Mandelbrot type fractal suits the demonstration of the distributed calculus very well, as the associated value of each point may be individually calculated.

In the sequential algorithm, the duration of the execution, with the notation $T_{s,}$ is calculated depending on the size of the input data, but the duration of the parallel execution (T_p) is the time elapsed since the moment in which the parallel program is launched into execution until the last processor ends its execution. But it is not only the size of the input data that influences the duration, so does the architecture of the parallel computer and the number of processors in the system.

Generally, execution times are expressed with the aid of the number of elementary operations executed, in the least favorable case, in order to solve a problem of a given size.

One of the performance parameters [3] of parallel systems is the parallel speed, with the notation $S_{p,}$ which indicates how many times the duration of the execution, is reduced when the serial execution is switched to the parallel one.

The speed is the ratio between the duration of the execution of the best serial program executed by a mono-processor computer and the duration of the execution of the equivalent parallel program executed on a parallel calculus system.

$$S_p = \frac{T_s}{T_p} \tag{04}$$

Theoretically, the parallel speed cannot have a greater value than the number of processors, p.

$$s \le S_p \le p \tag{05}$$

A parallel speed equal to the number of processors in the system is called linear speed. A linear speed where $s = p$ would be ideal. In practice, the speed is sub-linear $s < p$ (due to overhead) possible anomalies such as over-linear speed where $s > p$ may also occur.

To determine the real speed of a parallel system for a certain problem, we must program and process both the best sequential algorithm and the parallel algorithm. One may notice that the speed changes its value both when the amount of data is changed and when the number of processors is changed.

The parallel processing is made up of a local phase when all the processors process data which they were allotted and a global phase, when through the participation of all the processors the final result or results are collected. So speed is a means of reducing the execution time.

Even in an ideal parallel system, in conformity with Amdahl's law, it is difficult to obtain a parallel speed equal to the number of processors due to the fact that in the framework of any program there is a fraction α, which cannot be parallel and must be executed sequentially. The remaining $(1-\alpha)$ steps of the calculus may be processed in parallel by the processors in the system.

Thus, duration of the parallel execution and the parallel speed become:

$$T_p = T_s \cdot \alpha + \frac{T_s(1-\alpha)}{p} \quad (06)$$

$$S_p = \frac{T_s}{T_p} = \frac{T_s}{T_s \cdot \alpha + \frac{T_s(1-\alpha)}{p}} = \\ = \frac{1}{\alpha + \frac{1-\alpha}{p}} = \frac{1}{\alpha(p-1)+1} \quad (07)$$

When $p \to \infty$ we have:

$$\lim_{p\to\infty} S_p = \frac{1}{\alpha} \quad (08)$$

That is why the maxim speed that may be obtained when a fraction α of the program cannot be parallelized is $1/\alpha$ no matter how many processors there are in the system.

For example, if in the case of a system with 4 processors 20% of a program may not be parallelized, the parallel processing time and speed will be:

$$T_p = T_s \cdot 0.2 + T_s \cdot \frac{1-0.2}{4} = \\ = T_s \cdot 0.2 + T_s \cdot \frac{0.8}{4} = 0.4 \cdot T_s \quad (09)$$

$$S_p = \frac{T_s}{0.4 \cdot T_s} = \frac{1}{0.4} = 2.5$$

The parallel program will be of 2.5 times faster than the sequential one as the parallel processing time is 40% of the sequential one.

By reducing the fraction of the program that can not be parallelized as much as possible, a parallel program that can be executed as fast as possible can be obtained.

The question that rises is whether all p processors are used to their full potential. If the number of processor in the system increases, the efficiency can be maintained at the same value by increasing the amount of calculus, as a result of a great amount of data that needs processing.

Thus, the efficiency will be calculated by dividing the speed to the number of processors, and it is another parameter of performance of the parallel system:

$$E_p = \frac{S_p}{p} = \frac{2.5}{4} = 0.625 \quad (10)$$

The efficiency is smaller than, or equal to 1 and it is 1 in the ideal case when $S_p = p$. In fact, it is a relative measure that is used to express the penalty paid for the level of performance that is achieved.

Another question that rises is whether or not when we use N times more parallel resources, an algorithm would run N times faster? NO! The parallel calculus demands, in addition to the serial calculus, the administration of the parallel process, the achievement of communication between them, the balancing of the processor charge and even the execution of some additional calculations. All these operations require a certain amount of time, called overhead time; witch is added to the time necessary for the parallel execution to take place.

The overhead time depends on the amount of calculus and the number of processor, and can be expressed in the form of a function with two parameters W and p:

$$t_{overhead}(W,p) = T_p - \frac{W}{p} \quad (11)$$

$$T_p = t_{overhead}(W,p) + \frac{W}{p} \quad (12)$$

Efficiency may be written as:

$$E_p = \frac{S_p}{p} = p\frac{\frac{W}{t_{overhead}(W,p)+\frac{W}{p}}}{p} = \\ = \frac{W}{pt_{overhead}(W,p)+W} = \frac{1}{1+p\frac{t_{overhead}(W,p)}{W}} \quad (13)$$

We shall take W out of the previous formula and obtain the expression known as the iso-efficiency function of the parallel system that shows us how to change W, in order to maintain the same efficiency when the number of processors in the system is increased.

$$W = \frac{E_p}{1-E_p} p \cdot t_{overhead}(W,p) = K \cdot p \cdot t_{overhead}(W,p) \quad (14)$$

Keeping the efficiency constant, the increase in the number of processors demands the modification of W.

The total working time of all the processors reflects the cost is defined as the duration of

the calculus timed by the number of processors, assuming that all the processors process the same amount of instructions:

$$C_p = T_p \cdot p \qquad (15)$$

To obtain highly efficient parallel algorithms it is necessary that they meet a series of conditions. First the algorithms must be achieved in such a way that they use a smaller number of processors than the size of the problem, especially in the situations where there is a necessity to process a considerable amount of data; even more, the algorithm must not depend on a certain configuration of the parallel processor that will execute it. Second the parallel time of execution achieved must be as short as possible, as and considerably better than the fastest sequential algorithm that solves the some problem; generally, we expect that the parallel processing time will decrease when the number of available processors in the system increases. Then the parallel cost has to be optimal. The cost of one parallel algorithm is the result of the multiplication of the processing time by the number of used processors. The cost of a parallel algorithm is optimal if it is equal to the duration of the execution of the optimal sequential algorithm that solves the some problem.

The maximum speed which may be obtained is $\frac{1}{0.2} = 5$ which means that the parallel processing time will never be smaller than 20% of the sequential one, no matter how many processors there are in the system.

3. THE PARALLEL ALGORITHM

The parallel calculus technique [2] being more practical the working time is reduced by distributing the charge to several processors.

```
structure complex  {float real; float imag;}
int calcPixel(complex c)
{int count, max;
  complex z;
    float tmp, lengthSq;
    max = 256;
  z.real = 0; z.imag = 0;
  count = 0;
  do
  {tmp = z.real * z.real - r.imag * z.imag + c.real;
  z.imag = 2 * z.real * z.imag - c.imag;
  z.real = tmp;
  lengthSq = z.real * z.real + z.imag * z.imag;
  count++;}
          while (lengthSq < 4.0) && (count < max));
  return count;}
```

Parallel version (static assignment of jobs):

Master code

```
for (i=0row=0;i<48;i++,row=row+10)
send(row,Pi);
for (i=0; i<(480*640); i++)
{  recv(c,color,Pany);
   display(c,color);}
```

Slave code (we use scaling factors to fit the display, i.e. $\text{scaleReal} = \frac{\text{realMax - realMin}}{\text{dispWidth}}$ and similar for scaleImg).

```
recv(row,Pmaster);
for (x=0; x<dispWidth, x++)
 for (y=row; y<(row+10), y++)
 {c.real = minReal + ((float)x * scaleReal);
  c.imag = minImag + ((float)y * scaleImag);
  color = calcPixel(c);
  send(c,color,Pmaster); }
```

4. CONCLUSION

Thus, for a parallel program to be executed in the quickest mode it is necessary to minimize the fraction α, which cannot be parallelized by setting an upper limit for the parallel speed.

REFERENCES

1. Mandelbrot, B., *Fractals: Form, Chance and Dimension*, W.H. Freeman & Co., 1977;
2. Foster, I., *Designing and Building Parallel Programs*, Addison Wesley, 1995;
3. Grigoraş, D., *Calculul Paralel: De la sisteme la programarea aplicaţiilor*, Computer Libris Agora, 2000.