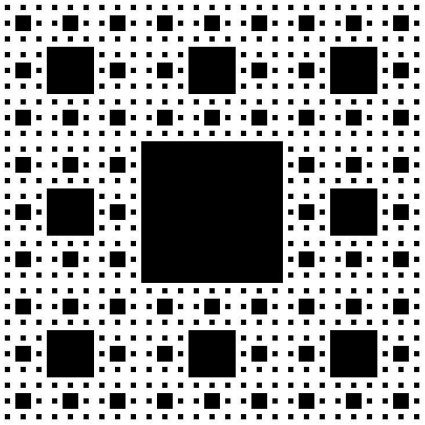

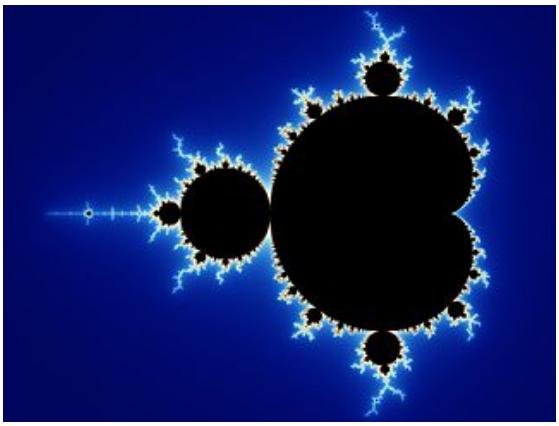

Abbildung 0.6: Left: Sierpinski-Fractal of fourth stage, Right: Mandelbrot set. Source: Wikipedia.

IWR, University of Heidelberg
Exercise Sheet 9

Winter term 2015/16
17. December 2015

Exercise for Course

Parallel High-Performance Computing

Dr. S. Lang

Return: 07. January 2016 at the beginning of the exercise or earlier

---

Task 20 New-Year task: Fractals with MPI (5 points)

In this task we want to calculate two fractals with MPI: The Sierpinski carpet and the Mandelbrot set. The task is highly-experimental and can be regarded as new-year goodie. Points are deliverd for all good ideas, hints, ...

Sierpinski Carpet

The first fractal is the so-called Sierpinsky carpet, see Figure 0.6 left. A Sierpinski carpet nth stage is generated in the following way:

- Start with a white square of side length l.
- Subdivide the square into 9 equal sized sub-squares.
- Color the mean square in black.
- Fill the other 8 sub-squares with Sierpinski carpets of stage $(n-1)$.

The Sierpinski carpet of 0th stage is a white square. On the homepage you find a sequential program, that generates a Sierpinski carpet of stage n. As parameters are passed the side length s of the intial square in pixels and the stage n, where should apply $s = k \cdot 3^n$, $n = 1, 2, \ldots$ and $k \in \mathbb{N}_+$, this means s is a multiple of some three power and the initial square allows a subdivision in n stages. The program generates a file `sierpinsky.ppm`, that you can display with almost any picture viewer.

Task

Extend the program such that it works in parallel with message passing (MPI), each process works only on a part of the area. Difficulties are during file output. In principle there are multiple possibilites:

1. Since the `ppm` files can get very large every processor could output its share in a dedicated file. Do not store all files, you have only about. 1 GB of disk space!
2. Test the parallel writing of files with the MPIfunctions `MPI_File_write()`, ...
3. You find a better format (regarding memory consumption) than `ppm`. In principle we only need black-white pictures and not the complete RGB color space.

Mandelbrot set

The Mandelbrot set (Figure 0.6, right) is defined by a complex sequence $f_c^{(n)}(z) := z_n^2 + c,\ z_n \in \mathbb{C}\ \forall n \in \mathbb{N}$, where c is a given complex number. If the sequence is bounded for the initial value $z_0 = 0$, then c is in the Mandelbrot set. It is known, that all c with $|c| > 2$ are not part of the Mandelbrot set. A picture of the Mandelbrot set can be displayed on the screen, where each pixel represents a point of the complex plain. Now n iterations will be calculated and the Mandelbrot points with $|f^{(n)}(c)| < 2$ black colored. Interessing are the boundary points of the Mandelbrot set. These can be colored in dependance of n, where n is smallest number for that holds $|f_c^{(n)}(n)| > 2$. The boundary can be investigated with increasing resolution. On the homepage you can find a sequential program, that contains the boundaries of the intervals and keeps the increments in the complex plain. It finally exports a `ppm` file.

Task

The program can be parallelized with the following strategy using message passing (MPI): The calculation of color values may be done for each pixel independently. Furthermore the number of iterations for each pixel varies under circumstances strongly. Thus the display is subdivided into several squares and one process controls as master. The slaves request from it after finishing their tasks new work (remember the TSP problem). We further use the fact, that all points in a square have the same color, if the boundary pixel also have the same color. With the initial square partitioning the slaves get now the task of calculating a coloring of a square assigned to them. Are the points on the boundary not all of the same color, the square is subdivided into four smaller ones and passed back to the master as new tasks. The depth of the nesting has of course some upper limit. The implementation is a little bit costly, therefore we consider a variant of Skjellum, Lusk and Gropp with parallel rendering of the output across a parallel X server.

1. Copy the directory `/export/home/phpc/phpc029/mandel` into your home directory.
2. Write into your `~/mpihosts` file just two lines, in each reads `localhost`. In this kind you can execute MPI-parallel programs on a multicore computer, thus the MIMD model is emulated without strict physical separation of the computers.
3. Now create a file `~/.bashrc` (or edit a already present file):

   ```
   1 PATH=/export/home/dune/dunekurs/mpich-install/bin:$PATH;
     export PATH
   ```

   The program only functions with `MPICH`. I have compiled a current version of MPICH locally in a folder and have adapted the Makefiles such that it is linked with an executable.
4. The new settings become present with `source ~/.bashrc`.
5. Compile the program `pmandel` by calling of `make` in the directory `mandel`.
6. Start the program by calling of

   ```
   1 mpiexec -f ~/mpihosts -n 2 ./pmandel
   ```

   in the directory `mandel`. Have fun!

After we have played a little bit, especially the following points are interesting from an implementational point of view:

- Dynamic distribution of the load by a master process,
- Usage of derived data types in MPI (methods `MPI_Type_contiguos, MPI_Type_commit` in `pm_genproc.c`),
- The usage of the graphics library MPE for visual contral of the calculations. There a parallel X11 is used.

After you have tested the program, look for documentation about the MPIdaten types and about the MPE library. Try to understand the interesting parts of the implementation. That's it!

! Watch out that you always log into a pool machine with the option `-X`: `ssh -X phlr...`, since otherwise there is no graphical output!

! After testing the Mandelbrot program revert the changes in the file `~/.bashrc` in any case, since you always would work with this MPICH library!

- Truely parallel Mandelbrot computing with parallel output does not work in the pool, since the graphical redirection had to be configured. (Eventually you discover, how that can be done as non-superuser?)
- You can of course compile your own version of MPICH locally in your home directory.