Università della Svizzera italiana | Institute of Computational Science ICS

High Performance Computing **2019**

Instructor: Prof. Olaf Schenk TA: Juraj Kardos, Radim Janalik, Aryan Eftekhari

Mini-project

Due date: 30 October 2019, 11:59pm

Parallel Programming with OpenMP

This assignment begins with the analysis of parallel programs, and will introduce you to parallel programming using OpenMP.

1. Parallel reduction operations using OpenMP *(30 Points)*

The file `dotProduct/dotProduct.cpp` in the git repository contains serial version of dot product of two vectors ($\alpha = a \cdot b, \quad \alpha \in \mathbb{R}, \{a, b\} \in \mathbb{R}^N$).

Solve the following problems:

1. Implement parallel version of dot product using OpenMP *reduction* clause.
2. Implement parallel version of dot product using OpenMP *parallel* and *critical* clauses.
3. Run the serial and parallel versions on icsmaster cluster, measuring their execution times. Make a performance scaling chart for serial version, parallel version using $1, 2, 4, 8$ threads working on vector lengths $100000, 1000000, 10000000, 100000000$.
4. In addition to performance scaling, plot the parallel efficiency.

Hint: You have to load `gcc` module.

2. Visualizing the Mandelbrot Set *(30 Points)*

Write a sequential code in C to visualize the Mandelbrot set. The set bears the name of the "Father of the Fractal Geometry," Benoît Mandelbrot. The Mandelbrot set is the set of complex numbers c for which the sequence $(z, z^2 + c, (z^2 + c)^2 + c, ((z^2 + c)^2 + c)^2 + c, (((z^2 + c)^2 + c)^2 + c)^2 + c, \ldots)$ does not approach infinity. Mandelbrot set images are made by sampling complex numbers and determining for each whether the result tends towards infinity when a particular mathematical operation is iterated on it. Treating the real and imaginary parts of each number as image coordinates, pixels are colored according to how rapidly the sequence diverges, if at all. More precisely, the Mandelbrot set is the set of values of c in the complex plane for which the orbit of 0 under iteration of the complex quadratic polynomial $z_{n+1} = z_n^2 + c$ remains bounded. That is, a complex number c is part of the Mandelbrot set if, when starting with $z_0 = 0$ and applying the iteration repeatedly, the absolute value of zn remains bounded however large n gets. For example, letting $c = 1$ gives the sequence $0, 1, 2, 5, 26, \ldots$ which tends to infinity. As this sequence is unbounded, 1 is not an element of the Mandelbrot set. On the other hand, $c = -1$ gives the sequence $0, -1, 0, -1, 0, \ldots$ which is bounded, and so -1 belongs to the Mandelbrot set.

The set is defined as follows:

$$\mathcal{M} := \{c \in \mathbb{C} : \text{the orbit } z, f_c(z), f_c^2(z), f_c^3(z), \ldots \text{ stays bounded}\}$$

where f_c is a complex function, usually $f_c(z) = z^2 + c$ with $z, c \in \mathbb{C}$. One can prove that if for a c once a point of the series $z, f_c(z), f_c^2(z), \ldots$ gets farther away from the origin than a distance of 2, the orbit will be unbounded, hence c does not belong to $\mathcal{M}$. Plotting the points whose orbits remain within the disk of radius 2 after `MAX_ITERS` iterations gives an approximation of the Mandelbrot set. Usually a color image is obtained by interpreting the number of iterations until the orbit "escapes" as a color value. This is done in the following pseudo code:

```
for all c in a certain range do
    z = 0
    n = 0
    while |c| < 2 and n < MAX_ITERS do
        z = z^2 + c
        n = n + 1
    end while
    plot n at position c
end for
```

The entire Mandelbrot set in Figure 1 is contained in the rectangle $-2.1 \leq \Re(c) \leq 0.7$, $-1.4 \leq \Im(c) \leq 1.4$. To create an image file, use the routines from `mandel/pngwriter.c` found on the git repository like so:

```c
#include "pngwriter.h"
png_data* pPng = png_create (width, height); // create the graphic

// plot a point at (x, y) in the color (r, g, b) (0 <= r, g, b < 256)
png_plot (pPng, x, y, r, g, b);

png_write (pPng, filename); // write to file
```

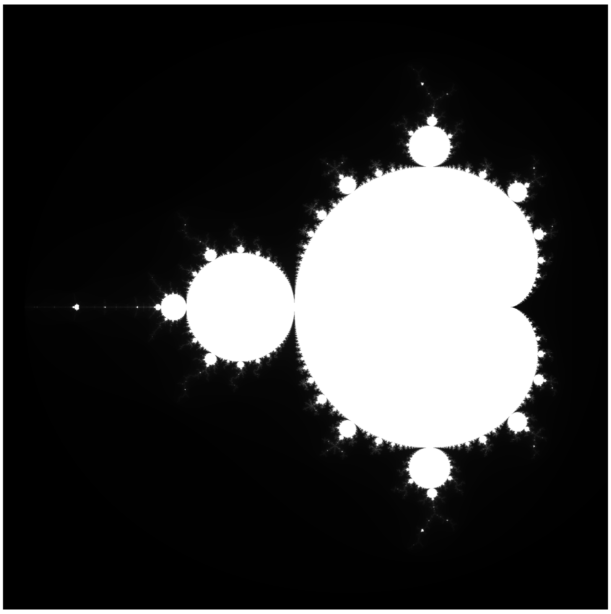

Figure 1. The Mandelbrot set

You need to link with `-lpng`. You can set the RBG color to white $(r, g, b) = (255, 255, 255)$ if the point at (x, y) belongs to the Mandelbrot set, otherwise it can be $(r, g, b) = (0, 0, 0)$

```
// plot the number of iterations at point (i, j)
int c = ((long) n * 255) / MAX_ITERS;
png_plot (pPng, i, j, c, c, c);
```

Record the time used to compute the Mandelbrot set. How many iterations could you perform per second? What is the performance in MFlop/s (assume that 1 iteration requires 8 floating point operations)? Try different image sizes. Please use the following C code fragment to report these statistics.

Solve the following problems:

1. Implement the computation kernel of the Mandelbrot set in the `mandel_seq.c`. Refer to the pseudo code above.
2. Count the total number of iterations in order to correctly compute the benchmark statistics. Use the variable `nTotalIterationsCount`.

Hints:
You have to load `gcc` module.
You have to compile on login node. On compute node you get an error "missing `png.h`".

3. Parallel Mandelbrot *(20 Points)*

Parallelize the code you have written using OpenMP. Compile the program using the GNU C compiler (`gcc`) with the option `-fopenmp`. Compare the timings of the parallelized program to those of the sequential program using a meaningful graphical representation.

Hints:

1. Copy `mandel_seq.c` to `mandel_par.c`
2. Open makefile and add target `mandel_par:`. You can copy the target `mandel_seq:`, change the filename and add `-fopenmp` flag.
3. Parallelize the algorithm in `mandel_par.c`

4. Bug Hunt *(20 Points)*

A number of short OpenMP programs are provided (`bugs/omp_bug1_1-5.c`), which all contain compile-time or run-time bugs. Identify the bugs, explain what is the problem and suggest how to fix it.
Hints:

1. check `tid`
2. check shared vs. private
3. check barrier
4. stacksize! `http://stackoverflow.com/questions/13264274/why-segmentation-fault-is-happening-in-this-openmp-code`
5. locking order?

Submission:

Submit the source code files in an archive file (tar, zip, etc) and show the TA the results. Furthermore, summarize your results and the obvervations for all exercises by writing an extended Latex summary. Use the Latex template from the webpage and upload the extended Latex summary as a PDF to iCorsi.