Problem set 1, Part 2

TDT4200, Fall 2016

Deadline: 2016–09–14, 20:00

Evaluation: Pass / Fail

Collaboration: This assignment must be done individually.

Delivery: It's Learning.
Deliver exactly two files via ItsLearning:

- *yourNTNUusername_ps1_part2.pdf*, with answers to the theory questions.
- *yourNTNUusername_code_ps1_part2.*{*zip* |*tar.gz* |*tar*} containing your solution to the programming tasks.

General notes: For details on servers to use, see It's Learning. Additional details/hints can be found in the recitation slides.

1 Theory

1.1 Caching

1. Explain the difference between a fully associative cache, a direct mapped cache and an n-way set associative cache.
2. According to Pacheco, programmers only have indirect control over caching. Why is this important? Give an example.
3. Give a brief description of the two main approaches to ensuring cache coherence.

1.2 Gustafson's law

Consider the following code:

```
for(int i = 0; i < problem_size*f; i++){
    function_a();
}
for(int i = 0; i < problem_size; i++){
    function_b();
}
```

Where `function_a()` is inherently serial, and `function_b()` can be parallelized fully without overhead. The serial execution times of `function_a()` and `function_b()` are the same.

1. The value of `f` can be 0 or 1. In which case does the program fit the assumptions of Amdahl's law and in which case the assumptions of Gustafson's law?

2. Calculate the speedup of the program using 2,4, and 8 processors and `problem_size`=5.
   - With `f` = 0, using Gustafson's law.
   - With `f` = 0, using Amdahl's law.
   - With `f` = 1, using Gustafson's law.
   - With `f` = 1, using Amdahl's law.
3. Explain why/why not you get the same results using Gustafson's and Amdahl's laws.

2 MPI Programming

You have been provided with the serial version of a code which computes the Mandelbrot set. It takes a single argument `n`, which determines whether an image should be written to disk (0 = no, 1 = yes). Most of the computation happens in the function `calculate`.
Any point on the grid can be represented as a complex number `c` where the real and imaginary parts can be interpreted as the `x` and `y`-coordinates respectively. `c` is a part of the Mandelbrot set if the absolute value of z_n remains less than 2. z_n is improved through iteration of the quadratic polynomial $z_{n+1}=z^2{}_n+c$. The algorithm is described in the following pseudo code:

1. Let z $\leftarrow$ c and i $\leftarrow$ 0. Define a maximum number of iterations MAX you want to perform.
2. If $\|z\| \geq 2$ or i = MAX then terminate. (Check implemented as $z_{im}^2 + z_{real}^2 \geq 4$)
3. Let z $\leftarrow$ z^2 + c
4. Let i $\leftarrow$ i + 1
5. Go to 2

In this part, your goal is to parallelize the function `calculate` in `mandel_mpi.c` using MPI. Currently, it is just a copy of the serial `mandel.c`

2.1 Mandatory

1. Parallelize the function `calculate` in `mandel_mpi.c` using MPI. The parallelized version should be properly load balanced.
2. Time your program, with P = {2,4,8} and scale the size of your problem by S = {1,2,3}. That is, `XSIZE`=S·2560, `YSIZE`=S·2048 and `MAXITER`=S·255. You can use a table to represent your results:

	s=1	s=2	s=3
p=2			
p=4			
p=8			

Example Table

For this exercise, timing should be done by modifying the code using `MPI_WTime()`. When timing the serial version, you should use the function `walltime()` which is provided in `mandel_c.c`.

2.2 Voluntary

Serialize ~ 50 % of the `calculate` function. How does this limit the potential speedup? Time the program.