TDDD56 Multicore and GPU computing
Lab 1: Load balancing

Nicolas Melot
nicolas.melot@liu.se

1st November 2012

1 Introduction

This laboratory aims to implement a parallel algorithm to generates a visual representation of the Mandelbrot set similar to Fig. 1. In such representation, the coordinates of a pixel depicts an element $c = (Cre, Cim) \in \mathbb{C}$. Its real part Cre denotes the position in the picture of the pixel along the x axis and the imaginary part Cim along the y axis. All black pixels represents an elements of $\mathbb{C}$ that belong to the Mandelbrot set. Although all pixels can be computed independently, the algorithm is embarrassingly parallel, implementing an efficient parallel version can be challenging, as some regions of the picture are more difficult to compute. This can easily result in computation load imbalance between threads, slowing down the whole program. In this lab, we illustrate such a situation and we implement load-balancing solution to utilize better parallel capabilities of multicore computers.

2 The Mandelbrot set and its representation

An element c of $\mathbb{C}$ is included in the Mandelbrot set if the norm of the Julia sequence $\|a_n\|$ shown in Eq. 1, starting with $z = (0,0)$, is bounded to $b \in \mathbb{R}^{+*}$. For every complex value $c \in \mathbb{C} \times ([Cre_{min} \cup Cre_{max}]$ and $[Cim_{min}, Cim_{max}])$ in the visual representation to be generated, the algorithm shown in Fig. 2 computes a sequence and counts the number of iterations before it detects divergence. If $MAXITER$ iterations was not enough to detect any, the algorithm returns $MAXITER + 1$, denoting that c is included in the Mandelbrot set. Due to the iterative characteristic of the algorithm, computing a point belonging to the Mandelbrot set (colored regions shown in Fig.1) can be significantly faster than computing a point included in the Mandelbrot set. In this latter case, The algorithm "gives up" after an arbitrarily-defined maximum iteration threshold.

$$\begin{cases} a_0 & = z \\ a_{n+1} & = a_n^2 + c, \forall n \in \mathbb{N}^* \end{cases} \tag{1}$$

3 Lab rooms

During the laboratory sessions, you have priority in rooms "Southfork" and "Konrad Suze" in the B building. See below for information about these rooms:

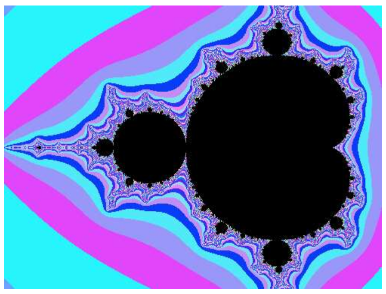

Figure 1: A representation of the Mandelbrot set (black area) in the range $-2 \leq Cre \leq 0.6$ and $-1 \leq Cim \leq 1$. The pixels in the black area require the full iteration number $MAXITER + 1$.

```
int
is_in_Mandelbrot(float Cre, float Cim)
{
  int iter;
  float x = 0.0, y = 0.0, xto2 = 0.0, yto2 = 0.0, dist2;

  for (iter = 0; iter <= MAXITER; iter++)
    {
      y = x * y;
      y = y + y + Cim;
      x = xto2 - yto2 + Cre;
      xto2 = x * x;
      yto2 = y * y;

      dist2 = xto2 + yto2;

      if ((int) dist2 >= 100)
        {
          break; // converges to infinity
        }
    }
  return iter;
}
```

Figure 2: C function to decide whether the complex number belongs to the Mandelbrot set, returning the number of iterations necessary to take the decision.

3.1 Konrad Suze (IDA)

- Intel®Xeon™ X5660[1]
  - 6 cores
  - 2.80GHz
- 6 GiB RAM
- OS: Debian Squeeze

During lab sessions, you are guaranteed to be alone using one computer at a time. Note that Konrad Suze is not accessible outside the lab sessions; you can nevertheless connect to one computer through ssh at *ssh <ida_student_id>@li21-<1..8>.ida.liu.se* using your *IDA* student id. You can use Octave to process your measurements and generate result graphs (see section 4).

3.2 Southfork (ISY)

- Intel®Core™ 2 Quad CPU Q9550[2]
  - 4 cores
  - 2.80GHz
- 4 GiB RAM
- OS: CentOS 6 i386

Southfork is open and accessible from 8:00 to 17:00 every day, except when other courses are taught. You can also remotely connect to *ssh <isy_student_id>ixtab.edu.isy.liu.se*, using your *ISY* student id. Southfork does not provide Octave to generate graphs but Matlab is available to run alternative scripts (see section 4).

4 Installation

Fetch the lab 1 skeleton source files from the CPU lab page *http://www.ida.liu.se/~nicme26/tddd56.en.shtml* and extract them to your personal storage space. You also need performance measurement and processing scripts available at http://www.ida.liu.se/~nicme26/measuring.en.shtml. Fetch the script source files and copy the content of the directory “Octave” (or “Matlab” if you work in Southfork) to the lab source directory.

5 Before the lab

Before coming to the lab, we recommend you do the following preparatory work

- Write a detailed explanation why computation load can be imbalanced and how it affects the global performance.
  Hint: What is necessary to compute a black point or a colored point?
- Describe a load-balancing method that would help reducing the performance loss due to load-imbalance

[1] http://ark.intel.com/products/47921/Intel-Xeon-Processor-X5660-(12M-Cache-2_80-GHz-6_40-GTs-Intel-QPI)
[2] http://ark.intel.com/products/33924/Intel-Core2-Quad-Processor-Q9550-(12M-Cache-2_83-GHz-1333-MHz-FSB)

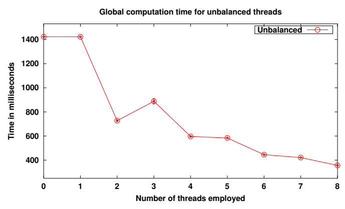

Figure 3: Performance of an unbalanced algorithm. The computation time does not lower accordingly with the increasing number of threads.

- Implement in advance all the algorithms required in sections 6 and 7 so you can measure their performance during lab sessions. Use the preprocessor symbol LOAD-BALANCE to take decision to use either no load-balancing or one or several load-balancing methods (the helper scripts assume a value of 0 for no load-balacing and 1 and 2 for two different load-balancing methods). This value is known at compile time and can be handled using preprocessor instructions such as *#if-#then-#else*; see skeleton for example.

6 During the lab session

Take profit of the exclusive access you have to the computer you use in the lab session to perform the following tasks

- Measure the performance of the naive parallel algorithm and generate a graph showing execution time as a function of number of threads involved.
  Hint: You will observe more easily the load-imbalance effects if you generate only pictures of the Mandelbrot set in the range Cre $\in [-2; +0.6]$ *and Cim* $\in [-1; +1]$. *We suggest to generate a picture of* 500×375 *pixels, using a maximum of* 256 *iterations. You are encouraged to change these parameters if this helps you to have a better understanding of the problem or to find a solution.*

- Measure the performance of the load-balanced variant and produce a graph featuring both load-balanced and load-imbalanced global execution time curves.

7 Lab demo

Demonstrate to your lab assistant the following elements:

1. Show the performance (execution time) as a function of number of threads measured on the naive parallel algorithm, through a clear diagram.

2. Explain the reason why some threads get more work than others

3. Explain the load-balancing strategy you implemented and argue why it helps improving performance

4. Show the performance of your load-balanced version and compare it to the naive parallel algorithm. Explain the reason of performance differences.

8 Helpers

The given source files are instrumented to measure the global and per thread execution time. Run *bash start compile* to compile several suggested variants, then *bash start run <name*

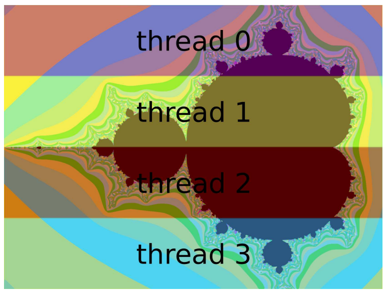

Figure 4: A possible partition for the computation of the representation of a Mandelbrot subset. Every thread receives an equally big subarea of the global picture to compute.

for experiment> to run them and measure their performance (note that you must name your experiment). Finally, run the script *plot_data.m* (*octave plot_data.m* or start Matlab and run the script plot_data.m) to generate graphs showing the behavior of your implementation. Observe that during performance measurement, the compile scripts define the symbol *LOADBALANCE* with values 0, 1 or 2. These values are meaningless if you don't use them is your source code, but they are intended to trigger alternative load-balancing methods (0 stands for no load-balancing). Modify the files *compile*, *run*, *variables* and *Makefile* at wish to fit further experiments you may need. You can read documentation about measuring performance and using the script set at the page *http://www.ida.liu.se/~nicme26/measuring.en.shtml*. Alternatively, feel free to use any other mean of measuring performance but in any case, make sure you can explain the measurement process and the numbers you show.

9 Investigate further

You can investigate further the load-imbalance issue when computing the Mandelbrot set. We suggest the following tasks

- *Good for exam*: Provide a performance analysis of the sequential Mandelbrot generator, using the big O notation. Analyze the naive parallel version and give speedup and efficiency.
- Analyze the load-balanced parallel variant using the big O notation. Compare with the naive parallel version analysis.
- *Just for fun*: Active the opengl implementation by appending GLUT=1 to the make command. Type 'h' to display controls. Compare smoothness of sequential, parallel naive and parallel load-balanced variants of the algorithm.
- *Just for fun*: Update the function *update_colors* (mandelbrot.c, line 210) to generate other fancy colors for the fractal.