University of Amsterdam
BSc Informatica
Dr Clemens Grelck

Concurrency en
Parallel Programmeren
2012/2013

Assignment 3

Directive-based Programming with OpenMP

Assignment 3.1: Wave equation simulation

Reconsider the 1-dimensional wave equation studied in assignments 1.1 and 2.1. Parallelize your sequential simulation code using OpenMP and repeat the experiments of assignment 1.1. Compare your results with those obtained by your own multithreaded code.

Experiment with different scheduling techniques and block sizes using 8 threads (on an 8-core cluster node). Report your results and explain why some schedulings perform better than others.

Assignment 3.2: Mandelbrot Set

On the reverse side you find a small program obtained from the internet which computes the Mandelbrot set (the code is also available on Blackboard in electronic form). Based on this sequential implementation, develop two parallel programs using OpenMP: one that produces the same output as the sequential code and one that dispenses with output, but still does all the computing. You are allowed to modify the code as you wish to make it more amenable to parallelisation.

Experimentally determine a suitable resolution `dxy` that results in convenient execution times: long enough to make timing accurate when running on up to 8 cores, but not longer than necessary. Augment the given code by timing facilities similar to those found in the wave equation framework; measure the complete program execution from beginning to end.

Report the best runtime for your chosen resolution that you can achieve on a DAS-4 cluster node for each of your codes (with and without output). Describe and motivate your solution with a short report.

Assignment due date: November 19, 2012, 12:00

```
// Usage: ./mandel > output
//   where "output" will be a binary image, 1 byte per pixel
//   see the fprintf towards the end for further processing instructions
//   and use the display command to show the image

// Define the range in x and y here:

const double yMin = -1.0;
const double yMax = +1.0;
const double xMin = -2.0;
const double xMax = +0.5;

// And here is the resolution:

const double dxy = 0.005;

#include <stdio.h>
#include <limits.h>

int main(void) {

    double cx, cy;
    double zx, zy, new_zx;
    unsigned char n;
    int nx, ny;

    // The Mandelbrot calculation is to iterate the equation
    // z = z*z + c, where z and c are complex numbers, z is initially
    // zero, and c is the coordinate of the point being tested. If
    // the magnitude of z remains less than 2 for ever, then the point
    // c is in the Mandelbrot set. We write out the number of iterations
    // before the magnitude of z exceeds 2, or UCHAR_MAX, whichever is
    // smaller.

    for (cy = yMin; cy < yMax; cy += dxy) {
        for (cx = xMin; cx < xMax; cx += dxy) {
            zx = 0.0;
            zy = 0.0;
            n = 0;
            while ((zx*zx + zy*zy < 4.0) && (n != UCHAR_MAX)) {
                new_zx = zx*zx - zy*zy + cx;
                zy = 2.0*zx*zy + cy;
                zx = new_zx;
                n++;
            }
            write (1, &n, sizeof(n)); // Write the result to stdout
        }
    }

    // Now calculate the image dimensions. We use exactly the same
    // for loops as above, to guard against any potential rounding errors.

    nx = 0;
    ny = 0;
    for (cx = xMin; cx < xMax; cx += dxy) {
        nx++;
    }
    for (cy = yMin; cy < yMax; cy += dxy) {
        ny++;
    }

    fprintf (stderr, "To␣process␣the␣image:␣"
                     "␣convert␣-depth␣8␣-size␣%dx%d␣gray:output␣out.jpg\n",
             nx, ny);
    return 0;
}
```