M. Troyer, P. Hadjidoukas
ETH Zentrum, ML H 44
CH-8092 Zürich

Fall semester 2016

Set 2 - Diffusion and Multithreading

Issued: October 7, 2016

Question 1: Diffusion in 2D

Heat flow in a medium can be described by the diffusion equation of the form

$$\frac{\partial \rho(\boldsymbol{r},t)}{\partial t} = D\nabla^2 \rho(\boldsymbol{r},t) \tag{1}$$

where $\rho(\mathrm{r},\mathrm{t})$ is a measure for the amount of heat at position $\boldsymbol{r}$ and time t and the diffusion coefficient D is constant. Lets define the domain Ω in two dimensions as $x, y \in [-1, 1]$. We will use open boundary conditions

$$\rho(x,y,t) = 0 \quad \forall\, t \geq 0 \,\mathrm{and}\, (x,y) \notin \Omega \tag{2}$$

and an initial density distribution

$$\rho(x,y,0) = \begin{cases} 1 & |x,y| < 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

a) Discretize equation (1) using forward Euler in time and central differences in space and write a serial code to model the time evolution of $\rho(x,y,t)$.
Hint: To run the code use the example parameters in Table 1.

Table 1: Example parameters.

	D	Ω	Δt
Set 1	1	128×128	0.00001
Set 2	1	256×256	0.000001
Set 3	1	1024×1024	0.00000001

b) Parallelize your code using manual C++ threads. Check that the parallel code produces the same result as the serial code and report your timings for $n = 1, 6, 12$ threads.
Hint: To run the code use the example parameters in Table 1.

c) Make a 2D density plot of $\rho(x,y,t)$ at $t = 0, 0.5, 1, 2$.

Question 2: Barrier - Synchronization with threading

A barrier is a synchronization point between multiple execution units. In this exercise we want to implement a `barrier` class using C++11 manual threading which fulfills the following syntax.

```
barrier b(nthreads);
// ... spawn 'nthreads' threads ...
// inside each thread:
b.wait()
```

The `b.wait()` statement returns only when all `nthreads` called that function.

a) Implement the `barrier` class and provide a small test for it.

b) Use the barrier in the diffusion code of Question 1 such that threads are kept alive and do not respawn on each iteration. Compare the timings with your previous implementation.