Reaction–Diffusion model

Team 8

Last revision: September 21, 2011

Outline

Project file structure

`project-rd/`

- `Makefile`: G++ and NVCC specific build instructions
- `main.cpp`: OpenGL and GLUT environment initialization and rendering
- `rd_kernel.cu`: GPU kernels and C `rd`-kernel wrapper
- `rd_kernel.h`: Prototype function for `rd`-kernel wrapper

PDEs

Two chemicals, denoted by concentrations U and V, which contain node point concentrations in 2D grid.

$$\frac{\delta U}{\delta t} = D_u \nabla^2(U) - UV^2 + F(1 - U)$$

$$\frac{\delta V}{\delta t} = D_v \nabla^2(V) - UV^2 - (F + k)V$$

PDEs

Two chemicals, denoted by concentrations U and V, which contain node point concentrations in 2D grid.

$$\frac{\delta U}{\delta t} = D_u \nabla^2(U) - UV^2 + F(1 - U)$$

$$\frac{\delta V}{\delta t} = D_v \nabla^2(V) - UV^2 - (F + k)V$$

Calculated on a cell-by-cell basis, grid indexed by i and j:

$$\frac{\delta u_{i,j}}{\delta t} = D_u \nabla^2(u_{i,j}) - u_{i,j} v_{i,j}^2 + F(1 - u_{i,j})$$

$$\frac{\delta v_{i,j}}{\delta t} = D_v \nabla^2(v_{i,j}) - u_{i,j} v_{i,j}^2 - (F + k)v_{i,j}$$

rd_kernel
Parent kernel


```
71  __global__ void rd_kernel(unsigned int width, unsigned int height,
72                            float dt, float dx, float Du, float Dv,
73                            float F, float k, float *U, float *V) {
74
75    // Coordinate of the current pixel (for this thread)
76    const uint2 co = make_uint2( blockIdx.x*blockDim.x + threadIdx.x,
77                                 blockIdx.y*blockDim.y + threadIdx.y );
78
79    // Linear index of the curernt pixel
80    const unsigned int idx = co.y*width + co.x;
81
82    if(co.y == 0 || co.y == height-1 || co.x == 0 || co.x == width-1) {
83      return;
84    }
85
86    // Load cell values from global memory
87    const float u = U[idx];
88    const float v = V[idx];
89
90    const float gradU = Du * laplacian_operator(U, dx, co, width, u)
91                        - u * v*v + F * (1.0f - u);
92
93    const float gradV = Dv * laplacian_operator(V, dx, co, width, v)
94                      + u * v*v - (F + k) * v;
95
96    // Eulerian integration
97    const float u_new = u + dt*gradU;
98    const float v_new = v + dt*gradV;
99
100   // Wait with write until all new values are computed
101   __syncthreads();
102
103   U[idx] = u_new;
104   V[idx] = v_new;
105 }
```

→Appendix

laplacian_operator
Helper kernel

$$\nabla^2 s_{i,j} = \frac{s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1} - 4s_{i,j}}{\Delta x^2}$$

```
56  // Calculate the Laplacian operator, nabla^2
57  __device__
58  float laplacian_operator(float *scalar_field, float ↩
        dx, uint2 co, unsigned int width, float val)
59  {
60    return (   val_sf(scalar_field, co.x+1, co.y, width)
61             + val_sf(scalar_field, co.x-1, co.y, width)
62             + val_sf(scalar_field, co.x, co.y+1, width)
63             + val_sf(scalar_field, co.x, co.y-1, width)
64             - 4.0*val )
65             / (dx * dx);
66  }
```

val_sf
Helper kernel


```
49  // Calculate linear index from 2D grid coordinates
        and return value from scalar field
50  __device__
51  float val_sf(float *sf, unsigned int x, unsigned int
        y, unsigned int width)
52  {
53    return sf[y*width + x];
54  }
```

Compiler flags

For debugging with CUDA-gdb:

```
nvcc -g -G -c $(INCLUDES) $< -o rd_kernel.cu
```

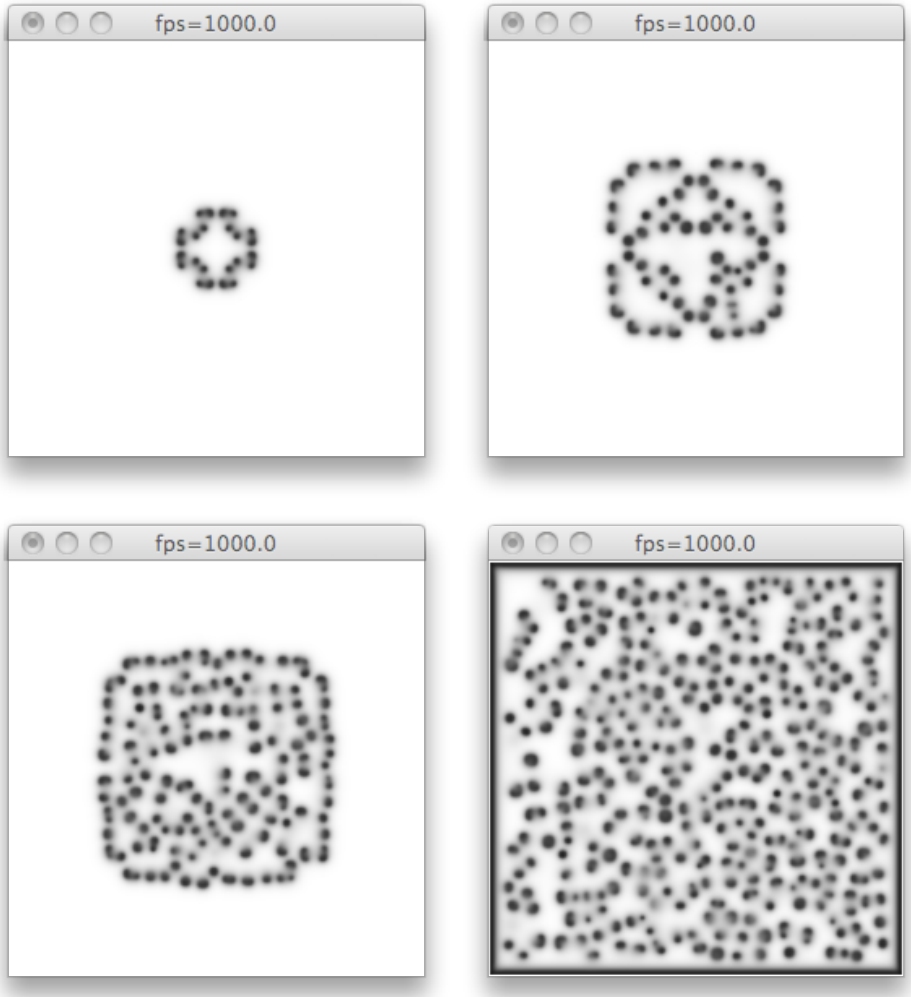
For final, fermi-optimized version:

```
nvcc -gencode arch=compute_20,code=sm_20 ↩
    -use_fast_math -Xcompiler -c ↩
    $(INCLUDES) $< -o rd_kernel.cu
```

Results

Testing platform: Quadro 4000 for Mac

— Fermi architecture, C.C. 4.0, 256 cuda cores



→Appendix

Discussion

For each cell, the values of the four axis-aligned neighbors are fetched when computing the laplacian operator ∇.

- In each time step, each value in the `U` and `V` arrays are fetched 5 times from global memory. These fetches are however *coalesced*.
- Shared memory as memory type for `U` and `V` arrays to avoid redundant data transfer.