


Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program of the Lobachevsky State University of Nizhni Novgorod among the world's research and education centers

Strategic initiative “Achieving leading positions in the field of supercomputer technology and high-performance computing”

Lobachevsky State University of Nizhni Novgorod

Faculty of Computational mathematics and cybernetics

Introduction to GPU programming

04 Practice
Numerical integration of heat equation

Bastrakov S.I.
Software department

Contents

- Problem statement
- Implementation on CPU
- Implementation on GPU

Problem statement

Problem statement

- Dirichlet problem for 2D heat equation

$$\frac{\partial u(x,y,t)}{\partial t} = \Delta u(x,y,t) + f(x,y,t)$$

$$a \le x \le b, \qquad c \le y \le d, \qquad 0 \le t \le T$$

$$u(x,y,0) = u_0(x,y)$$

$$u(a,y,t) = \varphi_1(y,t),\ u(b,y,t) = \varphi_2(y,t)$$

$$u(x,c,t) = \varphi_3(x,t),\ u(x,d,t) = \varphi_4(x,t)$$

Problem statement

- Uniform grid with steps Δx, Δy and Δt
- Denote $v^{(k)}_{ij}$ – grid value in point (x_i, y_j) at k-th time moment.
- We will use explicit scheme for numerical integration:

$$v^{(k+1)}_{ij} = v^{(k)}_{ij} + \Delta t * \left(f(x_i, y_j) + \frac{v^{(k)}_{i+1j} - 2v^{(k)}_{ij} + v^{(k)}_{i-1j}}{\Delta x^2} + \frac{v^{(k)}_{ij+1} - 2v^{(k)}_{ij} + v^{(k)}_{ij-1}}{\Delta y^2} \right)$$

Problem statement

- We will compute iteration by iteration in time.
- Computations within one time iteration are independent
- For simplicity assume boundary conditions are stationary (do not depend on time).
- In this case on each time step we can only update internal values.

Implementation on CPU

Implementation on CPU…

```
float f(float x, float y)
{
    return 0.05f * x * x + 0.001f * y * y * y;
}

// newV(i, j) = f(x, y) + v(i, j) +
//    dt * ((v(i + 1, j) - 2 * v(i, j) + v(i - 1, j)) / dx^2 +
//          (V(i, j + 1) - 2 * v(i, j) + v(i, j - 1)) / dy^2),
void step(int nx, int ny, float dx, float dy, float dt, float a, float c,
    const float * v, float * newV)
{
```

Implementation on CPU

```
    for (int i = 1; i < nx - 1; ++i)
        for (int j = 1; j < ny - 1; ++j)
        {
            float x = a + i * dx;
            float y = c + j * dy;
            newV[i * ny + j] = v[i * ny + j] + dt * (f(x, y) +
                (v[(i + 1) * ny + j] - 2.0f * v[i * ny + j] + v[(i - 1) * ny + j])
/ (dx * dx) +
                (v[i * ny + (j + 1)] - 2.0f * v[i * ny + j] + v[i * ny + (j - 1)])
/ (dy * dy));
        }
}
```

Implementation on GPU

Creating kernel…

```
__device__ float f_gpu(float x, float y)
{
    return 0.05f * x * x + 0.001f * y * y * y;
}

// newV(i, j) = f(x, y) + v(i, j) +
//    dt * ((v(i + 1, j) - 2 * v(i, j) + v(i - 1, j)) / dx^2 +
//          (v(i, j + 1) - 2 * v(i, j) + v(i, j - 1)) / dy^2),
__global__ void kernel(int nx, int ny, float dx, float dy, float dt,
float a,
    float c, const float * v, float * newV)
{
```

Creating kernel

```
    int i = 1 + blockIdx.x * blockDim.x + threadIdx.x;
    int j = 1 + blockIdx.y * blockDim.y + threadIdx.y;
    if ((i < nx - 1) && (j < ny - 1))
    {
        float x = a + i * dx;
        float y = c + j * dy;
        newV[i * ny + j] = v[i * ny + j] + dt * (f_gpu(x, y) +
            (v[(i + 1) * ny + j] - 2.0f * v[i * ny + j] + v[(i - 1) * ny + j]) /
(dx * dx) + (v[i * ny + (j + 1)] - 2.0f * v[i * ny + j] + v[i * ny + (j - 1)])
/ (dy * dy));
    }
}
```

Calling kernel

```
void step_gpu(int nx, int ny, float dx, float dy, float dt, float a,
float c,  const float * v_gpu, float * newV_gpu)
{
const int threadsX = 16;
    const int threadsY = 16;
    dim3 blocks(threadsX, threadsY);
    dim3 grid(((nx - 2) + (threadsX - 1)) / threadsX,
            ((ny - 2) + (threadsY - 1)) / threadsY);
    kernel<<<grid, blocks>>>(nx, ny, dx, dy, dt, a, c, v_gpu,
newV_gpu);
}
```

Function main…

```
int main()
{
    const float a = 0;
    const float b = 1;
    const float c = 2;
    const float d = 3;
    const int nx = 500;
    const int ny = 500;
    float dx = (b - a) / (float)(nx - 1);
    float dy = (d - c) / (float)(ny - 1);
```

Function main…

```
float dt = 1e-7f;
int num_time_steps = 100;
float * v = new float[nx * ny];
float * v_gpu;
cudaMalloc((void **) &v_gpu, nx * ny * sizeof(float));
float * newV = new float[nx * ny];
float * newV_gpu;
cudaMalloc((void **) &newV_gpu, nx * ny * sizeof(float));
float * v_verify = new float[nx * ny];
```

Function main…

```
    for (int i = 0; i < nx; ++i)
      for (int j = 0; j < ny; ++j)
      {
          v[i * ny + j] = 0.0f;
          newV[i * ny + j] = 0.0f;
      }
   cudaMemcpy(v_gpu, v, nx * ny * sizeof(float),
cudaMemcpyHostToDevice);
   cudaMemcpy(newV_gpu, newV, nx * ny * sizeof(float),
cudaMemcpyHostToDevice);
```

Function main…

```
for (int i = 0; i < num_time_steps; ++i)
{
    step(nx, ny, dx, dy, dt, a, c, v, newV);
    float * tmp = v;
    v = newV;
    newV = tmp;
}
```

Function main…

```
for (int i = 0; i < num_time_steps; ++i)
{
    step_gpu(nx, ny, dx, dy, dt, a, c, v_gpu, newV_gpu);
    cudaThreadSynchronize();
    float * tmp = v_gpu;
    v_gpu = newV_gpu;
    newV_gpu = tmp;
}
```

Function main…

```
 cudaMemcpy(v_verify, v_gpu, nx * ny * sizeof(float),
cudaMemcpyDeviceToHost);
    float difference = 0;
    for (int idx = 0; idx < nx * ny; ++idx)
        difference += (v[idx] - v_verify[idx]) * (v[idx] - v_verify[idx]);
    difference = sqrtf(difference);
    const float eps = 1e-5f;
    if (difference < eps)
        std::cout << "Test PASSED.\n";
    else
        std::cout << "Test FAILED.\n";
```

Function main

```
    cudaFree(v_gpu);
    delete [] v_verify;
    delete [] v;
    cudaFree(newV_gpu);
    delete [] newV;
    return 0;
}
```

Individual work

- Create implementation with 1D indexes.
- Implement boundary conditions.

References

- Sanders J., Kandrot E. CUDA by Example: An Introduction to General-Purpose GPU Programming. – Addison-Wesley Professional, 2010. – 312 p.
- NVIDIA CUDA C Programming Guide. [http://docs.nvidia.com/cuda/cuda-c-programming-guide/].

Authors

- Bastrakov S.I.,
  Assistant of the Software department of CMC faculty.
  bastrakov@vmk.unn.ru