CUDA Code

/* The list of relevant atom G_x has been already computed. It is stored in
* the val[] array. Also, the atom closest to the current point x has been
* stored at the first place in the array, i.e. val[0]. */
/* Similarly, the gradients and the hessians of the atoms functions have been
* already computed and stored in grad[] and hes[]. */
/* n -> number of relevant atoms */
/* SR -> solvent radius */
/* l, dl, ddl -> function l, its gradient and hessian */
/* f -> value of the function from the previous iteration */
/* g, dg, ddg -> i-th atom function, its gradient, its hessian */
/* k, dk, ddk -> dot product of gradients, its gradient and its hessian */
/* At the end, the final value of function l will be found in l */

for(int j = 0; j < 2; j++) {
    for(int i = 0; i < n; i++) {
        float f = l;  // Value of l from the previous iteration
        float g = val[i];  // Get a new atom from the list ...
        float dg = grad[i];  // ... its gradient
        float ddg = hes[i];  // ... and its hessian
        float k = dot(dl, dg);  // Dot product of the gradients
        float sink = sin(0.785 * k);
        float r_k = 1.0 - sink;  // Radius scaling function
        float r = SR * r_k;  // Scaled radius
        float r2 = r * r;
        float diff = (1 - g);
        float diff2 = diff * diff;

        if ((diff2 / r2) < 1.0) {  // Predicate of Eq. 6
            float q2 = 2 * r2 - diff2;
            float q = sqrt(q2);  // Square root part of Eq. 9
            float _q = 1.0/q;
            float _q2 = _q/_q2;

            // Update l (Eq. 9)
            l = 0.5*(2.0 * r + 1 + g - q);

            float3 dk = mul(ddl, dg) + mul(ddg, dl);  // Gradient of k (Eq. 12)
            float drdk = -0.785*SR*cos(0.785*k);  // 1st derivative of r wrt. k
            float drdkdk = 0.616225*SR*sin;  // 2nd derivative of r wrt. k

            float ddlf = 0.5*(1.0 + (diff)*_q);  // 1st derivative of l wrt. f
            float ddlg = 0.5*(1.0 - (diff)*_q);  // 1st derivative of l wrt. g
            float ddlr = 1.0 - (r) * _q;  // 1st derivative of l wrt. r
        }
    }
}
float dldfdf = r2 * _q2;  // 2nd derivative of l wrt. ff
float dldgdg = dldfdf;    // 2nd derivative of l wrt. gg
float dlrdrr = diff2 * _q2; // 2nd derivative of l wrt. rr
float dldfdg = -dldfdf;   // 2nd derivative of l wrt. fg
float dldfdr = -(diff * r * _q2); // 2nd derivative of l wrt. fr
float dlgdgr = -dldfdr;   // 2nd derivative of l wrt. gr

float3x3 J, JT;  // J and J^T in Eq. 13
fill3x3(J, dl, dg, drdk*dk);
fill3x3T(JT, dl, dg, drdk*dk);

float3x3 ddp;    // \dot H in Eq. 13
ddp.m[0] = make_float3(dldfdf, dldfdg, dldfdr);
ddp.m[1] = make_float3(dldfdg, dldgdg, dldgdg);
ddp.m[2] = make_float3(dldfdr, dldgdg, dldgdg);

float3x3 dk2;    // dk times dk^T in Eq. 13
dk2.m[0] = make_float3(dk.x*dk.x, dk.x*dk.y, dk.x*dk.z);
dk2.m[1] = make_float3(dk2.m[0].y, dk.y*dk.y, dk.y*dk.z);
dk2.m[2] = make_float3(dk2.m[0].z, dk2.m[1].z, dk.z*dk.z);

float3x3 ddk = (ddg * ddp) * 2.0;  // Approx. Hessian of k (Sec 4.2)

// Update the gradient of l (Eq. 11)
dl = dldf * dl + dldg * dg + (dlrd * drdk) * dk;

// Update the hessian of l (Eq. 13)
ddl = ddl*dldf + ddg*dldg + JT*(ddl_p*J) + dk2*(drdkdk*dlr) +
ddk*(drdk*dlr);