Chapter 4: GPU programming

Version 3: OpenCL

We can use shared memory in OpenCL exactly the same way as we do it in CUDA; we just need to replace `__shared__` with `__local`, use `barrier(CLK_LOCAL_MEM_FENCE)` for synchronization, and get used to the idea that in OpenCL documentation the concept is called "local memory":

```
const char* kernel_source =
R""(
__kernel void myppkernel(__global const float* r, __global float* d, int n, int nn) {
    int ja = get_local_id(0);
    int i = get_group_id(1);

    __global float* t = d + nn * nn;

    for (int jb = 0; jb < nn; jb += 64) {
        int j = jb + ja;
        float v = (i < n && j < n) ? r[n*i + j] : HUGE_VALF;
        d[nn*i + j] = v;
        t[nn*j + i] = v;
    }
}

__kernel void mykernel(__global float* r, __global const float* d, int n, int nn) {
    int ia = get_local_id(0);
    int ja = get_local_id(1);
    int ic = get_group_id(0);
    int jc = get_group_id(1);

    __global const float* t = d + nn * nn;

    __local float xx[4][64];
    __local float yy[4][64];

    float v[8][8];
    for (int ib = 0; ib < 8; ++ib) {
        for (int jb = 0; jb < 8; ++jb) {
            v[ib][jb] = HUGE_VALF;
        }
    }
    for (int ks = 0; ks < n; ks += 4) {
        int ija = ja * 8 + ia;
        int i = ic * 64 + ija;
        int j = jc * 64 + ija;
        for (int f = 0; f < 4; ++f) {
            int k = ks + f;
            xx[f][ija] = t[nn*k + i];
            yy[f][ija] = d[nn*k + j];
        }

        barrier(CLK_LOCAL_MEM_FENCE);

        #pragma unroll
        for (int f = 0; f < 4; ++f) {
            float y[8];
            for (int jb = 0; jb < 8; ++jb) {
                y[jb] = yy[f][jb * 8 + ja];
            }
            for (int ib = 0; ib < 8; ++ib) {
                float x = xx[f][ib * 8 + ia];
                for (int jb = 0; jb < 8; ++jb) {
                    v[ib][jb] = min(v[ib][jb], x + y[jb]);
                }
            }
        }

        barrier(CLK_LOCAL_MEM_FENCE);
    }
    for (int ib = 0; ib < 8; ++ib) {
        for (int jb = 0; jb < 8; ++jb) {
            int i = ic * 64 + ib * 8 + ia;
            int j = jc * 64 + jb * 8 + ja;
            if (i < n && j < n) {
                r[n*i + j] = v[ib][jb];
            }
        }
    }
}
)"";
```