```
#pragma OPENCL EXTENSION cl_arm_printf : enable
// game2048.cl
//
// OpenCL kernels to play the 2048 tile-sliding/growing game on parallel compute
//
// 1. slide() with locked[] all zeroed will tilt-bias the board toward slide_dir
// 2. coalesce() with the same array in unlocked[] "reacts" matching adjacents
// 3. slide() with the same array in locked[] does the final tilt-bias of a move
//
// the board is stored as a linear representation in row-major order, with zero-
// valued cells being "empty" such that a gravitational tilt will allow non-zero
// uphill tiles to slide into/past them
//
// for a global size of 4, game cells are indexed as follows:
//  0  1  2  3
//  4  5  6  7
//  8  9 10 11
// 12 13 14 15
// ...within board[], in global (by default) memory that all running kernels
// share although each works only within its row or column
//
// respectively left/right slides are represented as a slide_dir of -1 and +1,
// and up/down slides are represented as a slide_dir of -border-1 and +border+1

/*inline*/ void setInitialFinal(int *initial, int *final, int id, int n, int dir)
{
  *initial = id;
  if ((dir-1==0) || (dir+1==0)) { // slide within an nElement row
    *initial *= n;
  }
  if (dir > 0) {
    *final = *initial;
    *initial += dir*(n-1); // start at far end of row/col instead
  } else {
    *final = *initial - dir*(n-1); // adds a positive integer
  }
printf("workitem %d goes from %d to %d step %d", id, *initial, *final, -dir);
}

int slide(
  __global int* board, // 4x4 number grid etc.
  int nElements, // how many must be processed in the dimension of the slide
  int slide_dir) // direction of slide, one of: <-1(U), -1(L), +1(R), >+1(D)
{
  int iInitial, iFinal, i, j;
  int moves = 0;

  setInitialFinal(&iInitial, &iFinal, get_global_id(0), nElements, slide_dir);
  for (i = iInitial, j = iInitial; i != iFinal - slide_dir; i -= slide_dir)
    if (board[i]) {
      if (i != j) {
        board[j] = board[i];
        moves++;
      }
      j -= slide_dir;
    }
  for (; j != iFinal - slide_dir; j -= slide_dir)
    board[j] = 0;

  return moves;
}

int coalesce(
  __global int* board, // 4x4 number grid etc.
  int nElements, // how many must be processed in the dimension of the slide
  int slide_dir, // direction of slide, one of: <-1(U), -1(L), +1(R), >+1(D)
  int log_rep) // true if cell values stored as logarithms, false if linear
{
  int iInitial, iFinal, i, j;
  int repeatSlide = 0;

  setInitialFinal(&iInitial, &iFinal, get_global_id(0), nElements, slide_dir);

  j = iInitial+slide_dir; // move in lock step, always one behind i
  for (i = iInitial; i != iFinal - slide_dir; i -= slide_dir, j -= slide_dir)
    if (board[i]) {
      if ((i != iInitial) && (board[i] == board[j])) {
        board[j] = log_rep ? (board[j] + 1) : (board[j] << 1);
        board[i] = 0;
        repeatSlide++;
      }
    } else
      break; // reached end of nonzero values

  return repeatSlide;
}

__kernel void tilt(
  __global int* board, // 4x4 number grid etc.
  int nElements, // how many must be processed in the dimension of the slide
  int slide_dir, // direction of slide, one of: <-1(U), -1(L), +1(R), >+1(D)
  int log_rep, // true if cell values stored as logarithms, false if linear
  __global int* errarr) // zero if any change as a result of slide() or coalesce()
{
  int myError;

  myError = slide(board, nElements, slide_dir) ? 0 : 1; // >0 slid means no error

  if (coalesce(board, nElements, slide_dir, log_rep)) {
    myError = 0; // also not an error if any coalesced into a new tile
    slide(board, nElements, slide_dir);
  }

  errarr[get_global_id(0)] = myError;
}
```

```
// build with: gcc -o 2048 -L/usr/lib/path_to_libOpenCL.so playgame.c readkern.c -l
OpenCL

// renamed from convolution.c in ch. 4 of "Heterogeneous Computing with OpenCL"

#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>

#include <termios.h>            //termios, TCSANOW, ECHO, ICANON
#include <unistd.h>     //STDIN_FILENO


void termsetup(int newsetup){
  //http://stackoverflow.com/questions/1798511/how-to-avoid-press-enter-with-any-ge
tchar
  static struct termios oldt, newt;

  if (newsetup) {
    /*tcgetattr gets the parameters of the current terminal
    STDIN_FILENO will tell tcgetattr that it should write the settings
    of stdin to oldt*/
    tcgetattr( STDIN_FILENO, &oldt);
    /*now the settings will be copied*/
    newt = oldt;

    /*ICANON normally takes care that one line at a time will be processed
    that means it will return if it sees a "\n" or an EOF or an EOL*/
    newt.c_lflag &= ~(ICANON);

    /*Those new settings will be set to STDIN
    TCSANOW tells tcsetattr to change attributes immediately. */
    tcsetattr( STDIN_FILENO, TCSANOW, &newt);

  } else
    /*restore the old settings*/
    tcsetattr( STDIN_FILENO, TCSANOW, &oldt);
}

char* readSource(char*);
void chk(cl_int, const char*, cl_device_id*, cl_program*);

void printGrid(int* g, int Nx, int Ny) {
   int i, j, l;

   l = 0;
   for (j = 0; j < Ny; j++) {
     for (i = 0; i < Nx; i++) {
       int c = g[l++];
       if (c)
         printf("%6d ", c);
       else
         printf("[     ]");
     }
     printf("\n");
   }
   printf("\n");
}

int dropGrid(int* g, int NxNy, int v, int zer) {
  int i;
  const int zer_orig = zer;

  do
    for (i = 0; i < NxNy; i++)
      if (g[i] == 0)
        if (zer-- == 0) {
          g[i] = v;
          return i;
        }
  while (zer < zer_orig);

  return -1; // found no zeros, grid full
}

inline int ishoriz(int x) { return ((x+1 == 0) || (x-1 == 0)) ? 1 : 0; }

inline void done(int* grid){ free(grid); termsetup(0); exit(0); }

int main(int argc, char** argv) {
  int i/*, j, k, l*/;

   int up = 0, down = 0, left = 0, right = 0;

   // size of grid in x and y
   int xLog = 2, yLog = 2, xDim, yDim, errDim;
   if (argc > 2) {
     xLog = atoi(argv[2]);
     if (argc > 3) {
       yLog = atoi(argv[3]);
     }
   }
   xDim = 1<<xLog;
   yDim = 1<<yLog;
   errDim = (xDim > yDim) ? xDim : yDim;

   // Initialize the board position
   const size_t dataSize = xDim*yDim*sizeof(int);
   int* grid = (int*) calloc(xDim*yDim, sizeof(int));
   grid[1] = grid[xDim] = 2;
   printGrid(grid, xDim, yDim);
   termsetup(1);

   // Set up the OpenCL environment
   const cl_int SLIDE_UP = -xDim;
   const cl_int SLIDE_LF = -1;
   const cl_int SLIDE_RT = 1;
   const cl_int SLIDE_DN = xDim;
   const cl_int log_rep = 0;
   cl_int status;

   // Discover platform
   cl_platform_id platform;
   status = clGetPlatformIDs(1, &platform, NULL);
   chk(status, "clGetPlatformIDs", NULL, NULL);

   // Discover device
   cl_device_id device;
   cl_device_type pu = ((argc>1)?(('G'&*argv[1])=='G'):0) ? CL_DEVICE_TYPE_GPU
                                                          : CL_DEVICE_TYPE_CPU;
   printf("max workgroup size %d, requesting %cPU\n", (xDim>yDim)?xDim:yDim,
          (pu == CL_DEVICE_TYPE_GPU) ? 'G' : 'C');
   status = clGetDeviceIDs(platform, pu, 1, &device, NULL);
   chk(status, "clGetDeviceIDs", NULL, NULL);

   cl_uint numdims;
   status = clGetDeviceInfo(device, CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS,
                            sizeof(cl_uint), &numdims, NULL);
   chk(status, "clGetDeviceInfo", NULL, NULL);
   size_t dims[numdims];
   status = clGetDeviceInfo(device, CL_DEVICE_MAX_WORK_ITEM_SIZES,
                            numdims*sizeof(size_t), &dims, NULL);
   chk(status, "clGetDeviceInfo", NULL, NULL);
   printf("the max workgroup size of which is reported as %d\n", dims[0]);

   // Create context
   cl_context_properties props[3] = {CL_CONTEXT_PLATFORM,
       (cl_context_properties)(platform), 0};
```

```
   cl_context context;
   context = clCreateContext(props, 1, &device, NULL, NULL, &status);
   chk(status, "clCreateContext", NULL, NULL);

   // Create command queue
   // FIXME: "warning: ''clCreateCommandQueue'' is deprecated"
   cl_command_queue queue;
   queue = clCreateCommandQueue(context, device, 0, &status);
   chk(status, "clCreateCommandQueue", NULL, NULL);

   // Create a program object with source and build it
   const char* source = readSource("game2048.cl");
   cl_program program;
   program = clCreateProgramWithSource(context, 1, &source, NULL, &status);
   chk(status, "clCreateProgramWithSource", NULL, NULL);
   status = clBuildProgram(program, 1, &device, NULL, NULL, NULL);
   chk(status, "clBuildProgram", &device, &program);

   // Create the kernel objects and arguments
   // FIXME: is a cl_int at these addresses changeable? redo clSetKernelArg()?
   cl_kernel tilt = clCreateKernel(program, "tilt", &status);
   chk(status, "clCreateKernel", NULL, NULL);

   // Create space for the grid on the device
   // FIXME: can we use CL_MEM_USE_HOST_PTR to avoid clEnqueueWriteBuffer()?
   // FIXME: inefficient for getting a return value?
   cl_mem d_grid, d_invalid;
   d_grid = clCreateBuffer(context, 0, dataSize, NULL, &status);
   chk(status, "clCreateBuffer", NULL, NULL);
   d_invalid = clCreateBuffer(context, 0, errDim*sizeof(cl_int),
                              NULL, &status);
   chk(status, "clCreateBuffer", NULL, NULL);

   do {
     cl_int nElements, slide_dir;
     cl_int invalid[errDim];

     switch (getchar()) {
     case 'h':case 'a':case '4': left = 1; slide_dir = SLIDE_LF; nElements = xDim;
break;
     case 'j':case 's':case '2': down = 1; slide_dir = SLIDE_DN; nElements = yDim;
break;
     case 'k':case 'w':case '8': up = 1; slide_dir = SLIDE_UP; nElements = yDim; br
eak;
     case 'l':case 'd':case '6': right = 1; slide_dir = SLIDE_RT; nElements = xDim;
 break;
     case 'q':case '\033': done(grid);
     default : continue;  // applies to the do...while
     }
     putchar('\n');

     // set arguments each time?
     status  = clSetKernelArg(tilt, 0, sizeof(cl_mem), &d_grid);
     status |= clSetKernelArg(tilt, 1, sizeof(cl_int), &nElements);
     status |= clSetKernelArg(tilt, 2, sizeof(cl_int), &slide_dir);
     status |= clSetKernelArg(tilt, 3, sizeof(cl_int), &log_rep);
     status |= clSetKernelArg(tilt, 4, sizeof(cl_mem), &d_invalid);
     chk(status, "clSetKernelArg", NULL, NULL);

     // Copy inputs to the device
     status = clEnqueueWriteBuffer(queue, d_grid, CL_TRUE /*blocking_write*/,
                                   0 /*offset*/, dataSize, grid,
                                   0 /*events_in_ ...*/, NULL /*event_wait_list*/,
                                   NULL /*event*/);
     chk(status, "clEnqueueWriteBuffer", NULL, NULL);

     // Set the work item dimensions
     size_t globalSize[2] = {xDim, yDim};
     status = clEnqueueNDRangeKernel(queue, tilt, 1, NULL,
                                     globalSize + ishoriz(slide_dir), NULL,
                                     0, NULL, NULL);
     chk(status, "clEnqueueNDRangeKernel", NULL, NULL);

     status = clEnqueueReadBuffer(queue, d_grid, CL_TRUE /*blocking_read*/,
                                  0 /*offset*/, dataSize, grid,
                                  0 /*events_in_ ...*/, NULL /*event_wait_list*/,
                                  NULL /*event*/);
     chk(status, "clEnqueueReadBuffer", NULL, NULL);
     status = clEnqueueReadBuffer(queue, d_invalid, CL_TRUE /*blocking_read*/,
                                  0 /*offset*/, nElements*sizeof(cl_int), invalid,
                                  0 /*events_in_ ...*/, NULL /*event_wait_list*/,
                                  NULL /*event*/);
     chk(status, "clEnqueueReadBuffer", NULL, NULL);

     for (i = (nElements == xDim) ? yDim-1 : xDim-1; i >= 0; i--)
{
  printf("%d ", invalid[nElements]);
       if (!invalid[i])
         break; // found a valid move, so i will be >= 0
}
printf("\n");
     if (i >= 0) {
       dropGrid(grid, xDim*yDim, 2<<(1&random()), random()&((1<<(xLog+yLog))-1));
       printGrid(grid, xDim, yDim);
       up = down = left = right = 0;
     }
   } while (!(up && down && left && right));
}
```

```
// renamed from convolution.c in ch. 4 of "Heterogeneous Computing with OpenCL"

#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>

// This function reads in a text file and stores it as a char pointer
char* readSource(char* kernelPath) {

   cl_int status;
   FILE *fp;
   char *source;
   long int size;

   printf("Program file is: %s\n", kernelPath);

   fp = fopen(kernelPath, "rb");
   if(!fp) {
      printf("Could not open kernel file\n");
      exit(-1);
   }
   status = fseek(fp, 0, SEEK_END);
   if(status != 0) {
      printf("Error seeking to end of file\n");
      exit(-1);
   }
   size = ftell(fp);
   if(size < 0) {
      printf("Error getting file position\n");
      exit(-1);
   }

   rewind(fp);

   source = (char *)malloc(size + 1);

   int i;
   for (i = 0; i < size+1; i++) {
      source[i]='\0';
   }

   if(source == NULL) {
      printf("Error allocating space for the kernel source\n");
      exit(-1);
   }

   fread(source, 1, size, fp);
   source[size] = '\0';

   return source;
}

void chk(cl_int status, const char* cmd, cl_device_id* dev, cl_program* program) {

   if(status != CL_SUCCESS) {
      printf("%s failed (%d)\n", cmd, status);

      // from buildProgramDebug.c found in a blog at http://dhruba.name

      if (program && dev) {
         // build failed
         char* programLog;
         size_t logSize;

         // check build error and build status first
         clGetProgramBuildInfo(*program, *dev, CL_PROGRAM_BUILD_STATUS,
                               sizeof(cl_build_status), &status, NULL);

         // check build log
         clGetProgramBuildInfo(*program, *dev,
                               CL_PROGRAM_BUILD_LOG, 0, NULL, &logSize);
         programLog = (char*) calloc (logSize+1, sizeof(char));
         clGetProgramBuildInfo(*program, *dev,
                               CL_PROGRAM_BUILD_LOG, logSize+1, programLog, NULL);
         printf("Build failed; status=%d, programLog:nn%s",
                status, programLog);
         free(programLog);
      }

      exit(-1);
   }
}
```