OpenCL alapok

Az OpenCL

- Adat- és feladat párhuzamos modell
- Az ISO C99 szabvány részhalmaza
  - párhuzamos kiegészítésekkel
- Numerikus műveletek az IEEE754 alapján
- Beágyazott és mobil eszközök támogatása
- OpenGL, OpenGL ES adatcsere támogatás

Az OpenCL

- Heterogén platform támogatás
  - Párhuzamos CPU-k
  - Grafikus hardver (GPU)
  - Jelfeldolgozó processzorok (DSP)
  - Cell/B.E. processzor

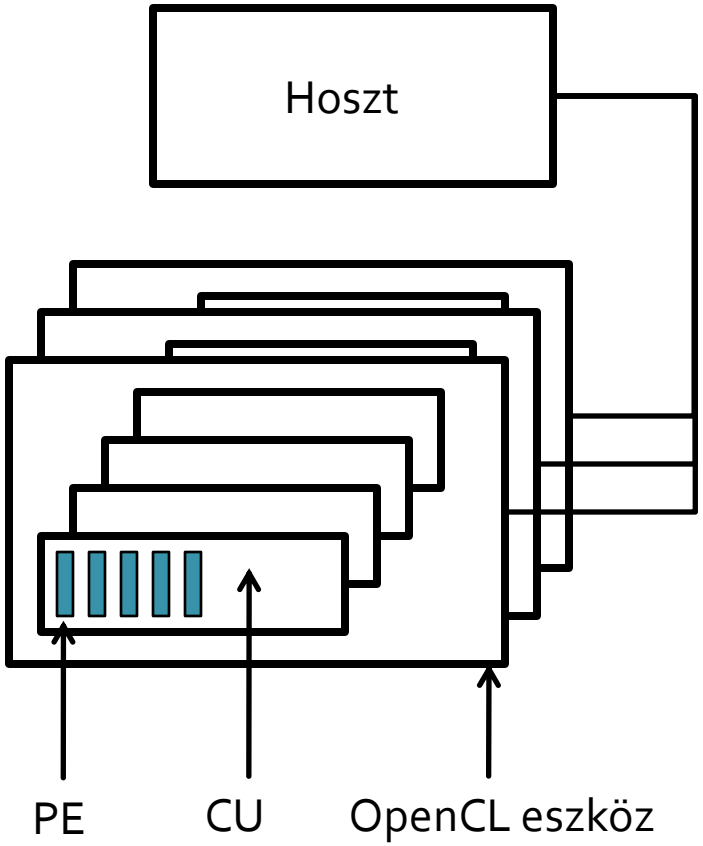
OpenCL architektúra

- Az OpenCL elemei
  - Platform modell
  - Végrehajtási séma
  - Memória modell
  - Program modell

Platform modell

- Hoszt eszköz
- OpenCL eszközök

- Számító egységek (CU)
  - Feldolgozó egységek (PE)
    - SIMD (közös utasítás számláló)
    - SPMD (saját utasítás számláló)



Végrehajtási séma

- Hoszt program
  - Kontextus management
  - Végrehajtás vezérlés
- Kernel program
  - Számító egységek vezérlése

Végrehajtási séma

- Kernel program
  - Index tér (NDRange)
  - Munkacsoportok (work-groups)
  - Feladat egységek (work-items)
    - Globális azonosító (global ID)
    - Azonos program a munkacsoportban
    - A vezérlés eltérhet egységenként

Végrehajtási séma

- Kernel program
  - Index tér (NDRange)
  - Munkacsoportok (work-groups)
    - Finomabb index felbontás
    - Munkacsoport azonosító (work-group ID)
    - A feladat egységeknek lokális azonosító (local ID)
  - Feladat egységek (work-items)

Végrehajtási séma

- Kernel program
  - Index tér (NDRange)
    - N dimenziós problématér (N=1,2,3)
    - Minden cimke egyforma dimenziójú
  - Címzések
    - Globális címtér: (G_x, G_y)
    - Munkacsoport méret: (S_x, S_y)
      - Munkacsoport azonosító: (w_x, w_y)
    - Lokális azonosító: (s_x, s_y)

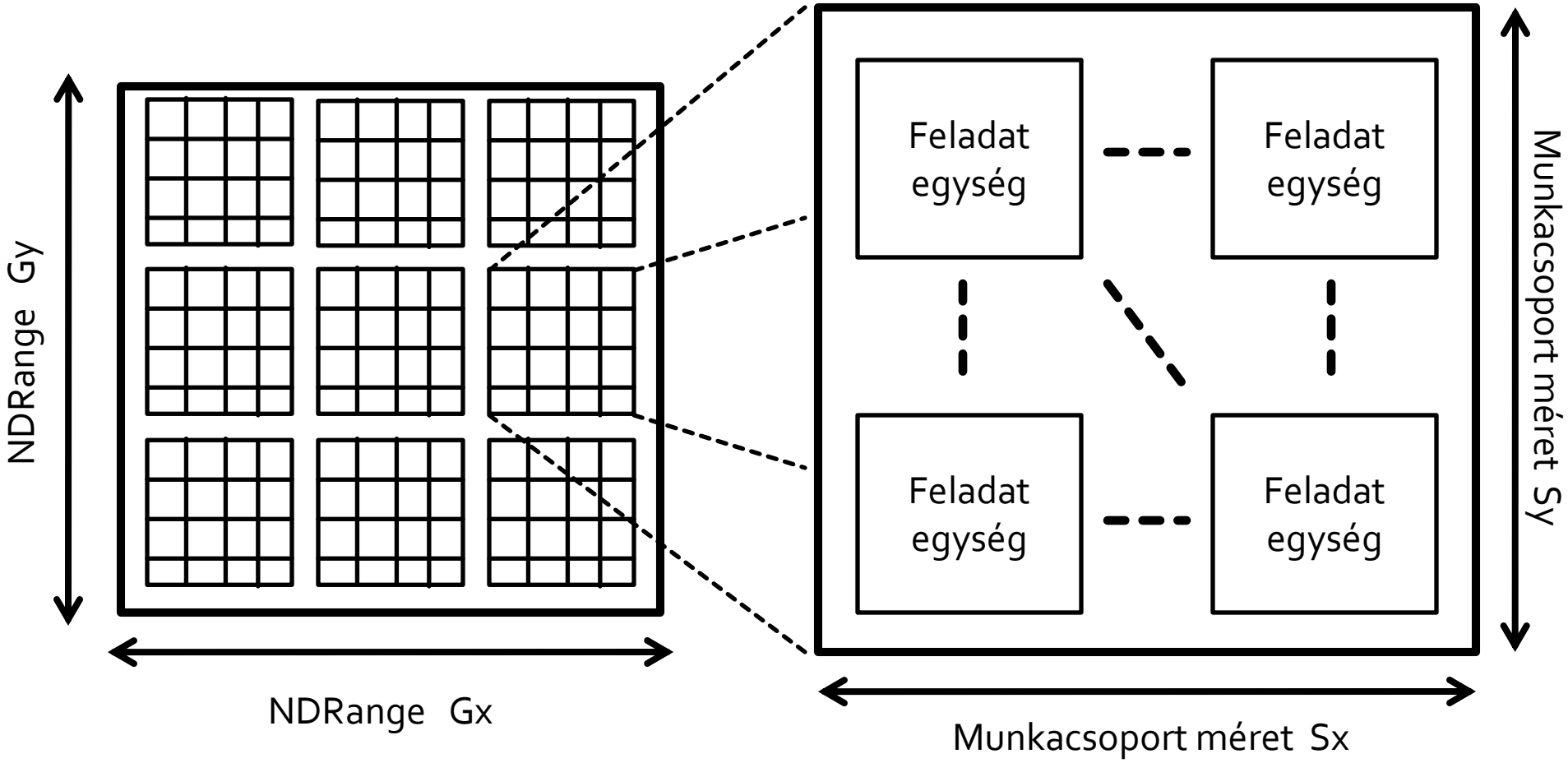
Végrehajtási séma

- Azonosítók számítása
  - Globális címtér: (G_x, G_y)
  - Munkacsoport méret: (S_x, S_y)
    - Munkacsoport azonosító: (w_x, w_y)
  - Lokális azonosító: (s_x, s_y)

  - Globális azonosító: $(g_x, g_y) = (w_x \cdot S_x + s_x, w_y \cdot S_y + s_y)$
  - Munkacsoportok száma: $(W_x, W_y) = (G_x / S_x, G_y / S_y)$
  - Lokális azonosító: $(w_x, w_y) = ((g_x - s_x) / S_x, (g_y - s_y) / S_y)$

Végrehajtási séma

- Azonosítók számítása



Végrehajtási séma

- Kontextus (context)
  - Eszközök: OpenCL eszközök halmaza
  - Kernelek: OpenCL függvények csoportja
  - Program objektumok:
    - Kernel forráskód
    - Végrehajtható bináris reprezentáció
  - Memória objektumok:
    - A hoszt és OpenCL eszközök által használt memória
    - A kernelek által látott értékek

Végrehajtási séma

- Parancs sorok (command-queue)
  - A hoszt által ellenörzött parancs folyam
  - A kernelek végrehajtását vezérli
  - Parancsok
    - Kernel végrehajtás
    - Memória műveletek
    - Szinkronizáció

Végrehajtási séma

- Parancs sor módok
  - In-order végrehajtás
    - A parancsok fifo módon hajtódnak végre
    - Soros végrehajtása a parancs sornak
  - Out-of-order végrehajtás
    - A parancsok nem várják meg az előző befejeződését
    - Explicit szinkronizáció szükséges

Végrehajtási séma

- Kernel típusok
  - OpenCL kernel
    - OpenCL C függvények
    - Az OpenCL eszközön futtathatóak
  - Natív kernel
    - A hoszton futó függvények
    - A memória objektumokat megosztottan használhatja
    - Opcionális

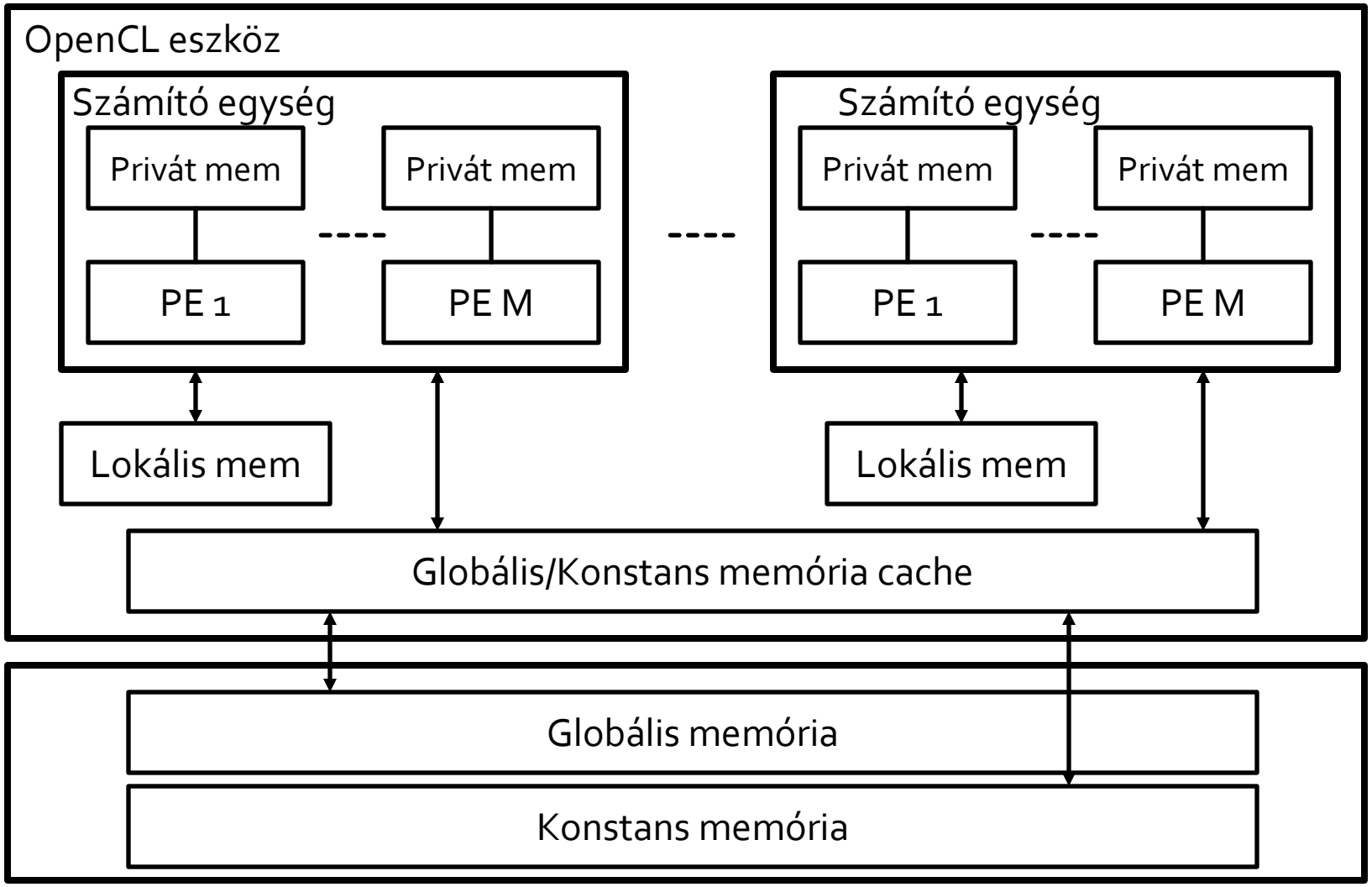
Memória modell

- Négy használható memória régió
  - Globális memória
    - Írható/olvasható a feladat egységekből
    - Bármely eleme elérhető bármelyik PE-ből
    - A hoszt allokálja
  - Konstans memória
    - Olvasható a feladat egységekből
    - A hoszt allokálja és tölti fel értékekkel
    - A kernelben statikusan is definiálható

Memória modell

- Négy használható memória régió
  - Lokális memória
    - A munkacsoport osztott memóriája
    - Minden feladat egység írhatja/olvashatja a munkacsoportban
    - A hoszt nem fér hozzá
  - Privát memória
    - A feladat egység saját írható/olvasható memória területe
    - Csak az adott feladat egység látja
    - A hoszt nem fér hozzá

Memória modell



Memória modell

- A globális memória területet a hoszt kezeli
  - Memória foglalás
  - Memória másolás
    - Szinkron és aszinkron másolás
  - Memória felszabadítás

  - A globális memória leképezhető a hoszt memóriába

Memória modell

- Relaxált konzisztencia
  - Nincs garantált konzisztencia a feladat egységek között
  - Feladat egységen konzisztens olvasás/írás

  - Lokális memória
    - Munkacsoporton konzisztens
  - Globális memória
    - Munkacsoporton konzisztens
    - Munkacsoportok között nincs garantált konzisztencia

  - Parancsok között szinkronizációval kényszeríthető ki

Program modell

- Adat párhuzamos modell
  - Párhuzamos művelet halmaz
  - Adat – feladat egység összerendelés
  - Nincs korlátozva egy-egy kapcsolatra

  - Hierarchikus adat párhuzamosság
    - Explicit felbontás
      - Teljes probléma tér és felbontása munkacsoportokra
    - Implicit felbontás
      - Teljes probléma tér és automatikus felbontás

Program modell

- Feladat párhuzamos modell
  - A kernel egyetlen példányban
  - Index tértől független

  - Műveletek vektor típusokon
  - Több független feladat
  - Natív kernel tetszőleges párhuzamosítással

Szinkronizáció

- Munkacsoport szinkronizáció
  - A feladat egységek szinkronizációja
  - work-group barrier
    - blokkoló hívás
    - minden egységnek a barrier-re kell futnia
  - Munkacsoportok között nincs szinkronizáció

Szinkronizáció

- Parancs sor szinkronizáció
  - command-queue barrier
    - Garantálja a barrier előtti parancsok végrehajtódását
    - A barrier utáni parancs számára konzisztens memóriát biztosít
    - Parancs sorok között nincs szinkronizáció
  - Várakozás eseményre
    - Minden parancs generál egy eseményt ha végrehajtódott
    - Egy parancs végrehajtását várakoztathatjuk egy esemény bekövetkeztéig

OpenCL C

- Skalár típusok
  - bool
  - unsigned char, char (8 bites egész)
  - unsigned short, short (16 bites egész)
  - unsigned int, int (32 bites egész)
  - unsigned long, long (64 bites egész)
  - float (IEEE754 lebegőpontos)
  - half (16 bites float)
  - size_t (sizeof operátor típusa 32/64 bites)
  - ptrdiff_t (két pointer különbsége 32/64 bites)
  - (u)intptr_t (pointer típus)
  - void

OpenCL C

- Vektor típusok
  - (u)char*n*
  - (u)short*n*
  - (u)int*n*
  - (u)long*n*
  - float*n*

  - Az előjeles változat kettes komplemens
  - Az u az előjel nélküliséget jelöli
  - n lehet 2,4,8,16

OpenCL C

- Vektor komponensek
  - Swizzle operátor (.xyzw)
    - float4 f; f.xy; f.xxyy;
  - Numerikus indexek (.s[0-9|a-f|A-F])
    - float4 f; f.s12;
    - float16; f.saBcdE
  - Felezés (.odd, .even, .lo, .hi)
    - float4 f; f.hi; f.even.lo;
    - float4 left, right;

      float8 interleaved;

      interleaved.even = left; interleaved.odd = right;

OpenCL C

- Konverziók típusok között
  - Implicit konverzió
    - Korlátozott mértékben használható
    - Skalár típusok között
  - Explicit konverzió
    - Skalár – vektor konverzió
      - float4 f = (float4)1.0;
    - Vektor típusok közötti konverzió
      - destType*n* convert_destType_sat_roundingMode(sourceType*n*)
        - _sat - értékkészletre vágás
        - _roundingMode – kerekítés
      - uchar4 u; int4 c = convert_int4(u);

OpenCL C

- Konverziók típusok között
  - Azonos méretű típusok között
  - as_type*n*()
    - float f = 1.0f;

      uint u = as_uint(f); // 0x3f800000 lesz az értéke
    - float4 f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);

      int4 i = as_int4(f);

      // (0x3f800000, 0x40000000, 0x40400000, 0x40800000)

OpenCL C

- Memória terület jelölők
  - __global : globális memória
    - __global float4 color;
  - __local : lokális memória
    - __local float16 shared;
  - __contant : konstans memória
    - __constant float uniformData;
    - hoszt oldalról inicializálható
  - __private : privát memória
    - __private float8 workItemExclusive;

OpenCL C

- Függvény jelölők
  - __kernel : OpenCL függvény
    - Csak OpenCL eszközön hajtható végre
    - A hoszt program meghívhatja
    - Más OpenCL kernel meghívhatja
  - __attribute__ : fordító segítő attribútumok
    - vec_type_hint(type*n*) : vektor műveletek mérete
      - A feladat egységek összevonhatóak a fordító álltal

OpenCL C

- Beépített függvények
  - Feladat egység információk
    - uint get_work_dim()
    - size_t get_global_size(uint dimIdx);
    - size_t get_global_id(uint dimIdx);
    - size_t get_local_size(uint dimIdx);
    - size_t get_local_id(uint dimIdx);
    - size_t get_num_groups(uint dimIdx);
    - size_t get_group_id(uint dimIdx);

OpenCL C

- Beépített függvények
  - Matematikai függvények
    - float, half, egész típusokon
  - Általánosan használt függvények
    - float típusokon
  - Geometriai függvények
    - float típusokon
  - Összehasonlító függvények
    - float típusokon
    - pl. isequal(float, float)

      isfinite(float)

OpenCL C

- Beépített függvények
  - Vektor betöltő függvények
    - pointer – vektor konverzió
  - Vektor sorosító függvények
    - vektor – pointer konverzió

OpenCL C

- Beépített függvények
  - Szinkronizációs függvények
    - barrier(flag);
      - CLK_LOCAL_MEM_FENCE : lokális memóriát konzisztensé teszi
      - CLK_GLOBAL_MEM_FENCE : globális memóriát konzisztensé teszi
    - mem_fence(flag);
    - read_mem_fence(flag);
    - write_mem_fence(flag);

OpenCL C

- Beépített függvények
  - Aszinkron memória másolás
    - Globális memóriából lokális memóriába
    - Lokális memóriából globális memóriába
    - event_t async_work_group_copy(…);
    - wait_group_events(…, eventList);
  - Prefetch
    - A globális memória egy részének előtöltése a cache-be

Első OpenCL programom

```
#include <iostream>
#include <CL/opencl.h>

#define DATA_SIZE (1024*1240)

int main(int argc, char* argv[]){
  cl_int err;

  size_t global;  // globális problématér
  size_t local;   // lokális problématér

  cl_platform_id platform;
  err = clGetPlatformIDs(1, &platform, NULL);
  if(err != CL_SUCCESS){
    std::cerr << "Error: Failed to find a platform!" << std::endl;
    return EXIT_FAILURE;
  }

//...
```

Első OpenCL programom

```
  cl_device_id device_id;
  err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device_id, NULL);
  if(err != CL_SUCCESS){
    std::cerr << "Error: Failed to create a device group!" << std::endl;
    return EXIT_FAILURE;
  }

  cl_context context;
  context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);
  if (!context) {
    std::cerr << "Error: Failed to create a compute context!" << std::endl;
    return EXIT_FAILURE;
  }

  cl_command_queue commands;
  commands = clCreateCommandQueue(context, device_id, 0, &err);
  if (!commands) {
    std::cerr << "Error: Failed to create a command commands!" << std::endl;
    return EXIT_FAILURE;
  }
// ...
```

Első OpenCL programom

```
  cl_program program;
  program = clCreateProgramWithSource(context, 1,
                                (const char **) &KernelSource, NULL, &err);
  if (!program) {
    std::cerr << "Error: Failed to create compute program!" << std::endl;
    return EXIT_FAILURE;
  }

  err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
  if (err != CL_SUCCESS) {
    size_t len;
    char buffer[2048];

    std::cerr << "Error: Failed to build program executable!" << std::endl;
    clGetProgramBuildInfo(program, device_id, CL_PROGRAM_BUILD_LOG,
                          sizeof(buffer), buffer, &len);
    std::cerr << buffer << std::endl;
    exit(1);
  }

// ...
```

Első OpenCL programom

```
  cl_kernel kernel;
  kernel = clCreateKernel(program, "square", &err);
  if (!kernel || err != CL_SUCCESS) {
    std::cerr << "Error: Failed to create compute kernel!" << std::endl;
    exit(1);
  }

  float* data = new float[DATA_SIZE];    // bemenő adathalmaz
  float* results = new float[DATA_SIZE]; // eredmény halmaz
  unsigned int correct;
  cl_mem input;                          // device memória a bemenetnek
  cl_mem output;                         // device memória az eredménynek

  // A bemenet véletlen számok halmaza
  unsigned int count = DATA_SIZE;
  for(int i = 0; i < count; i++){
    data[i] = rand() / (float)RAND_MAX;
  }

// ...
```

Első OpenCL programom

```
  input = clCreateBuffer(context,  CL_MEM_READ_ONLY, sizeof(float) * count,
                                                                  NULL, NULL);
  output = clCreateBuffer(context, CL_MEM_WRITE_ONLY,sizeof(float) * count,
                                                                  NULL, NULL);
  if (!input || !output) {
    std::cerr << "Error: Failed to allocate device memory!" << std::endl;
    exit(1);
  }

  // Bemenő adat másolása az eszköz globális memóriájába
  err = clEnqueueWriteBuffer(commands, input,
                             CL_TRUE, 0, sizeof(float) * count,
                             data, 0, NULL, NULL);
  if (err != CL_SUCCESS) {
    std::cerr << "Error: Failed to write to source array!" << std::endl;
    exit(1);
  }

// ...
```

Első OpenCL programom

```
  // Kernel paraméterek beállítása
  err = 0;
  err  = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
  err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
  err |= clSetKernelArg(kernel, 2, sizeof(unsigned int), &count);
  if (err != CL_SUCCESS) {
    std::cerr << "Error: Failed to set kernel arguments! " << err << std::endl;
    exit(1);
  }

  // Munkacsoport méret meghatározása
  err = clGetKernelWorkGroupInfo(kernel, device_id,
                                 CL_KERNEL_WORK_GROUP_SIZE,
                                 sizeof(local), &local, NULL);
  if (err != CL_SUCCESS) {
    std::cerr << "Error: Failed to retrieve kernel work group info! "
              <<  err << std::endl;
    exit(1);
  }
// ...
```

Első OpenCL programom

```
  // Kernel indítása
  global = count;
  err = clEnqueueNDRangeKernel(commands, kernel,
                               1, NULL, &global, &local,
                               0, NULL, NULL);
  if (err) {
    std::cerr << "Error: Failed to execute kernel!" << std::endl;
    return EXIT_FAILURE;
  }

  // Az eredmény bevárása
  clFinish(commands);
  // Az eredmény visszaolvasása
  err = clEnqueueReadBuffer( commands, output,
                             CL_TRUE, 0, sizeof(float) * count,
                             results, 0, NULL, NULL );
  if (err != CL_SUCCESS) {
    std::cerr << "Error: Failed to read output array! " <<  err << std::endl;
    exit(1);
  }
// ...
```

Első OpenCL programom

```
  // Az eredmény ellenőrzése
  correct = 0;
  for(int i = 0; i < count; i++) {
    if(results[i] == data[i] * data[i])
      correct++;
  }

  std::cout << "Computed " << correct << "/" <<
               count << " correct values" << std::endl;
  std::cout << "Computed " << 100.f * (float)correct/(float)count <<
               "% correct values" << std::endl;

// ...
```

Első OpenCL programom

```
  // Takarítás
  delete [] data; delete [] results;

  clReleaseMemObject(input);
  clReleaseMemObject(output);
  clReleaseProgram(program);
  clReleaseKernel(kernel);
  clReleaseCommandQueue(commands);
  clReleaseContext(context);

  return 0;

}
```

Első OpenCL programom

- OpenCL kernel

```
__kernel void square(
                      __global float* input,
                      __global float* output,
                     const unsigned int count){
  int i = get_global_id(0);
  if(i < count){
    output[i] = input[i] * input[i];
  }
}
```