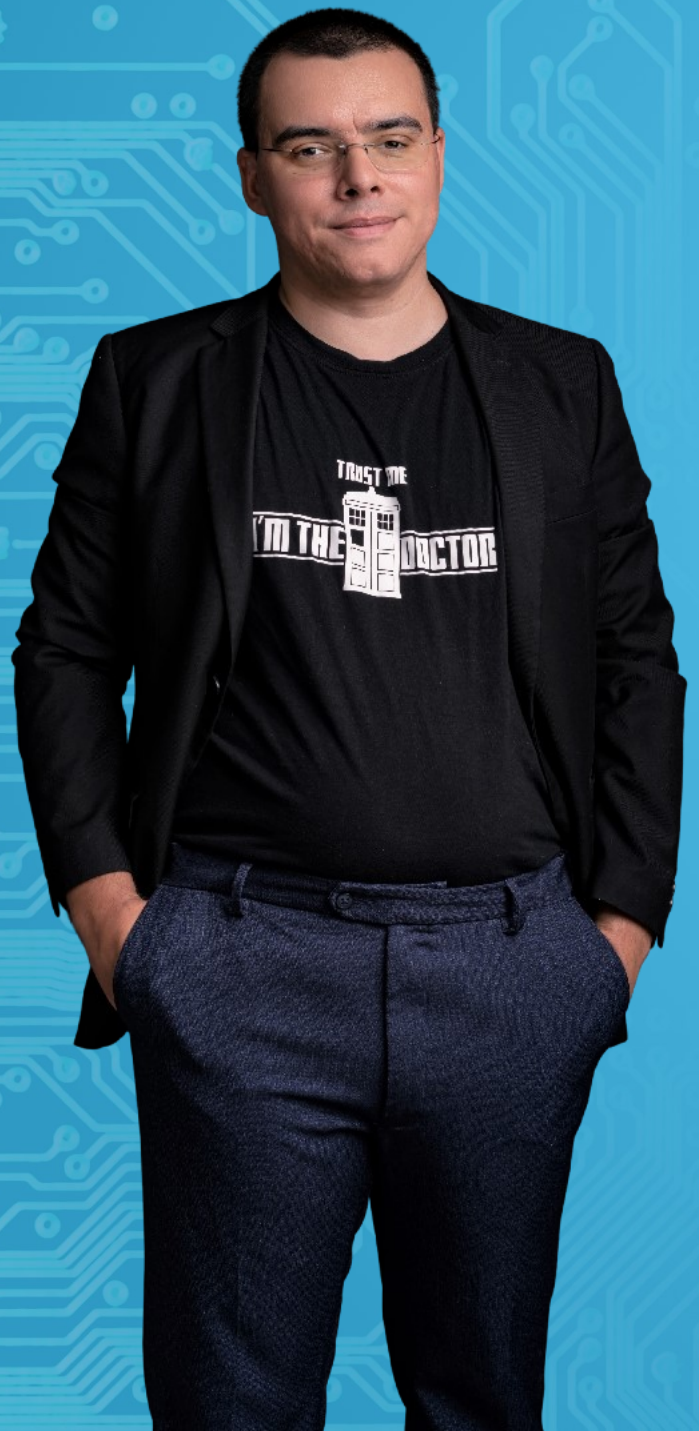

Arhitecturi Paralele
Introducere GPU

Lect. Dr. Ing. Cristian Chilipirea – cristian.chilipirea@mta.ro

Descoperirea platformelor (AMD/Intel/NVIDIA/...)

```
cl_int clGetPlatformIDs(↓ cl_uint num_entries,
                        ↑ cl_platform_id* platforms,
                        ↑ cl_uint* num_platforms);
```

Descoperirea platformelor

```
cl_int clGetPlatformIDs(↓ cl_uint num_entries,
                        ↑ cl_platform_id* platforms,
                        ↑ cl_uint* num_platforms);
```

Întoarce numărul de platforme existente pe sistem.

Descoperirea platformelor

```
cl_int clGetPlatformIDs(↓ cl_uint num_entries,
                        ↑ cl_platform_id* platforms,
                        ↑ cl_uint* num_platforms);
```

Întoarce id-urile platformelor în vectorul prealocat.

Descoperirea platformelor

```
cl_int clGetPlatformIDs(↓ cl_uint num_entries,
                        ↑ cl_platform_id* platforms,
                        ↑ cl_uint* num_platforms);
```

Mărimea vectorului platforms.

Descoperirea platformelor

```
cl_int clGetPlatformIDs(↓ cl_uint num_entries,
                        ↑ cl_platform_id* platforms,
                        ↑ cl_uint* num_platforms);
```

Întoarce CL_SUCCESS sau cod de eroare.

Decoperirea informațiilor despre platformă

```
cl_int clGetPlatformInfo(↓ cl_platform_id platform,
                         ↓ cl_platform_info param_name,
                         ↓ size_t param_value_size,
                         ↑ void* param_value,
                         ↑ size_t* param_value_size_ret);
```

Decoperirea informațiilor despre platformă

```
cl_int clGetPlatformInfo(↓ cl_platform_id platform,
                         ↓ cl_platform_info param_name,
                         ↓ size_t param_value_size,
                         ↑ void* param_value,
                         ↑ size_t* param_value_size_ret);
```

ID-ul platformei pentru care se doresc informații.

Decoperirea informațiilor despre platformă

```
cl_int clGetPlatformInfo(↓ cl_platform_id platform,
                         ↓ cl_platform_info param_name,
                         ↓ size_t param_value_size,
                         ↑ void* param_value,
                         ↑ size_t* param_value_size_ret);
```

Ce tip de informații se cer. Ex: CL_PLATFORM_NAME

Decoperirea informațiilor despre platformă

```
cl_int clGetPlatformInfo(↓ cl_platform_id platform,
                         ↓ cl_platform_info param_name,
                         ↓ size_t param_value_size,
                         ↑ void* param_value,
                         ↑ size_t* param_value_size_ret);
```

Informația certură este întoarsă într-un vector prealocat.

Decoperirea informațiilor despre platformă

```
cl_int clGetPlatformInfo(↓ cl_platform_id platform,
                         ↓ cl_platform_info param_name,
                         ↓ size_t param_value_size,
                         ↑ void* param_value,
                         ↑ size_t* param_value_size_ret);
```

Mărimea vectorului.

Decoperirea informațiilor despre platformă

```
cl_int clGetPlatformInfo(↓ cl_platform_id platform,
                         ↓ cl_platform_info param_name,
                         ↓ size_t param_value_size,
                         ↑ void* param_value,
                         ↑ size_t* param_value_size_ret);
```

Mărimea informației returnate (Ex: Nr de caractere).

Decoperirea informațiilor despre platformă

```
cl_int clGetPlatformInfo(↓ cl_platform_id platform,
                         ↓ cl_platform_info param_name,
                         ↓ size_t param_value_size,
                         ↑ void* param_value,
                         ↑ size_t* param_value_size_ret);
```

Întoarce CL_SUCCESS sau cod de eroare.

Descoperirea dispozitivelor (CPU/GPU efectiv)

```
cl_int clGetDeviceIDs(↓ cl_platform_id platform,
                      ↓ cl_device_type device_type,
                      ↓ cl_uint num_entries,
                      ↑ cl_device_id* devices,
                      ↑ cl_uint* num_devices);
```

Descoperirea dispozitivelor

```
cl_int clGetDeviceIDs(↓ cl_platform_id platform,
                      ↓ cl_device_type device_type,
                      ↓ cl_uint num_entries,
                      ↑ cl_device_id* devices,
                      ↑ cl_uint* num_devices);
```

Platforma pentru care se caută dispozitivele.

Descoperirea dispozitivelor

```
cl_int clGetDeviceIDs(↓ cl_platform_id platform,
                      ↓ cl_device_type device_type,
                      ↓ cl_uint num_entries,
                      ↑ cl_device_id* devices,
                      ↑ cl_uint* num_devices);
```

Tipul dispozitivului Ex: CL_DEVICE_TYPE_CPU, CL_DEVICE_TYPE_GPU, CL_DEVICE_TYPE_ALL

Descoperirea dispozitivelor

```
cl_int clGetDeviceIDs(↓ cl_platform_id platform,
                      ↓ cl_device_type device_type,
                      ↓ cl_uint num_entries,
                      ↑ cl_device_id* devices,
                      ↑ cl_uint* num_devices);
```

Este întoarsă lista de ID-uri dispozitiv într-un vector prealocat.

Descoperirea dispozitivelor

```
cl_int clGetDeviceIDs(↓ cl_platform_id platform,
                      ↓ cl_device_type device_type,
                      ↓ cl_uint num_entries,
                      ↑ cl_device_id* devices,
                      ↑ cl_uint* num_devices);
```

Mărimea vectorului.

Descoperirea dispozitivelor

```
cl_int clGetDeviceIDs(↓ cl_platform_id platform,
                      ↓ cl_device_type device_type,
                      ↓ cl_uint num_entries,
                      ↑ cl_device_id* devices,
                      ↑ cl_uint* num_devices);
```

Numărul de dispositive.

Descoperirea dispozitivelor

```
cl_int clGetDeviceIDs(↓ cl_platform_id platform,
                      ↓ cl_device_type device_type,
                      ↓ cl_uint num_entries,
                      ↑ cl_device_id* devices,
                      ↑ cl_uint* num_devices);
```

Întoarce CL_SUCCESS sau cod de eroare.

Decoperirea informațiilor despre dispozitiv

```
cl_int clGetDeviceInfo(↓ cl_device_id device,
                       ↓ cl_device_info param_name,
                       ↓ size_t param_value_size,
                       ↑ void* param_value,
                       ↑ size_t* param_value_size_ret);
```

Decoperirea informațiilor despre dispozitiv

```
cl_int clGetDeviceInfo(↓ cl_device_id device,
                       ↓ cl_device_info param_name,
                       ↓ size_t param_value_size,
                       ↑ void* param_value,
                       ↑ size_t* param_value_size_ret);
```

ID-ul dispozitivului pentru care se doresc informații.

Decoperirea informațiilor despre dispozitiv

```
cl_int clGetDeviceInfo(↓ cl_device_id device,
                       ↓ cl_device_info param_name,
                       ↓ size_t param_value_size,
                       ↑ void* param_value,
                       ↑ size_t* param_value_size_ret);
```

Ce tip de informații se cer. Ex: CL_DEVICE_NAME

Decoperirea informațiilor despre dispozitiv

```
cl_int clGetDeviceInfo(↓ cl_device_id device,
                       ↓ cl_device_info param_name,
                       ↓ size_t param_value_size,
                       ↑ void* param_value,
                       ↑ size_t* param_value_size_ret);
```

Informația certură este întoarsă într-un vector prealocat.

Decoperirea informațiilor despre dispozitiv

```
cl_int clGetDeviceInfo(↓ cl_device_id device,
                       ↓ cl_device_info param_name,
                       ↓ size_t param_value_size,
                       ↑ void* param_value,
                       ↑ size_t* param_value_size_ret);
```

Mărimea vectorului.

Decoperirea informațiilor despre dispozitiv

```
cl_int clGetDeviceInfo(↓ cl_device_id device,
                       ↓ cl_device_info param_name,
                       ↓ size_t param_value_size,
                       ↑ void* param_value,
                       ↑ size_t* param_value_size_ret);
```

Mărimea informației returnate (Ex: Nr de caractere).

Decoperirea informațiilor despre dispozitiv

```
cl_int clGetDeviceInfo(↓ cl_device_id device,
                       ↓ cl_device_info param_name,
                       ↓ size_t param_value_size,
                       ↑ void* param_value,
                       ↑ size_t* param_value_size_ret);
```

Întoarce CL_SUCCESS sau cod de eroare.

Creare context (permite lucru cu memoria device-urilor ca și cum ar fi doar un device)

```
cl_context clCreateContext(const cl_context_properties* properties,
        cl_uint num_devices,
        const cl_device_id* devices,
        void (CL_CALLBACK* pfn_notify)(const char* errinfo,
                const void* private_info,
                size_t cb,
                void* user_data),
        void* user_data,
        cl_int* errcode_ret);
```

Creare context

```
cl_context clCreateContext(const cl_context_properties* properties,
        cl_uint num_devices,
        const cl_device_id* devices,
        void (CL_CALLBACK* pfn_notify)(const char* errinfo,
                const void* private_info,
                size_t cb,
                void* user_data),
        void* user_data,
        cl_int* errcode_ret);
```

Se poate selecta proprietăți gen care platformă să fie folosită.
Pentru noi: 0.

Creare context

```
cl_context clCreateContext(const cl_context_properties* properties,
        cl_uint num_devices,
        const cl_device_id* devices,
        void (CL_CALLBACK* pfn_notify)(const char* errinfo,
                const void* private_info,
                size_t cb,
                void* user_data),
        void* user_data,
        cl_int* errcode_ret);
```

Vector cu ID-uri de dispozitive pentru care se creează contextul.

Creare context

```
cl_context clCreateContext(const cl_context_properties* properties,
        cl_uint num_devices,
        const cl_device_id* devices,
        void (CL_CALLBACK* pfn_notify)(const char* errinfo,
                const void* private_info,
                size_t cb,
                void* user_data),
        void* user_data,
        cl_int* errcode_ret);
```

Numărul de dispozitive.

Creare context

```
cl_context clCreateContext(const cl_context_properties* properties,
        cl_uint num_devices,
        const cl_device_id* devices,
        void (CL_CALLBACK* pfn_notify)(const char* errinfo,
                const void* private_info,
                size_t cb,
                void* user_data),
        void* user_data,
        cl_int* errcode_ret);
```

Funcție callback care va fi apelată pentru a raporta erori din context.
Pentru noi: NULL

Creare context

```
cl_context clCreateContext(const cl_context_properties* properties,
        cl_uint num_devices,
        const cl_device_id* devices,
        void (CL_CALLBACK* pfn_notify)(const char* errinfo,
                const void* private_info,
                size_t cb,
                void* user_data),
        void* user_data,
        cl_int* errcode_ret);
```

Date cu care va fi apelată funcția de callback.
Pentru noi: NULL.

Creare context

```
cl_context clCreateContext(const cl_context_properties* properties,
        cl_uint num_devices,
        const cl_device_id* devices,
        void (CL_CALLBACK* pfn_notify)(const char* errinfo,
                const void* private_info,
                size_t cb,
                void* user_data),
        void* user_data,
        cl_int* errcode_ret);
```

Întoarce CL_SUCCESS sau cod de eroare. Trebuie prealocat.

Creare context

```
cl_context clCreateContext(const cl_context_properties* properties,
        cl_uint num_devices,
        const cl_device_id* devices,
        void (CL_CALLBACK* pfn_notify)(const char* errinfo,
                const void* private_info,
                size_t cb,
                void* user_data),
        void* user_data,
        cl_int* errcode_ret);
```

Contextul este returnat pentru a putea folosi altor apeluri OpenCL.

Creare coadă de comenzi (pot fi multiple)

```
cl_command_queue clCreateCommandQueueWithProperties(
        ↓ cl_context context,
        ↓ cl_device_id device,
        ↓ cl_command_queue_properties properties,
        ↑  cl_int* errcode_ret);
```

Creare coadă de comenzi

```
cl_command_queue clCreateCommandQueueWithProperties(
        ↓ cl_context context,
        ↓ cl_device_id device,
        ↓ cl_command_queue_properties properties,
        ↑  cl_int* errcode_ret);
```

Context valid.

Creare coadă de comenzi

```
cl_command_queue clCreateCommandQueueWithProperties(
        ↓ cl_context context,
        ↓ cl_device_id device,
        ↓ cl_command_queue_properties properties,
        ↑  cl_int* errcode_ret);
```

Dispozitiv ce face parte din context.

Creare coadă de comenzi

```
cl_command_queue clCreateCommandQueueWithProperties(
        ↓ cl_context context,
        ↓ cl_device_id device,
        ↓ cl_command_queue_properties properties,
        ↑  cl_int* errcode_ret);
```

Proprietăți gen execuție în ordine aleatorie, profiling, mărime coadă.

Pentru noi: 0

Creare coadă de comenzi

```
cl_command_queue clCreateCommandQueueWithProperties(
        ↓ cl_context context,
        ↓ cl_device_id device,
        ↓ cl_command_queue_properties properties,
        ↑ cl_int* errcode_ret);
```

Întoarce CL_SUCCESS sau cod de eroare. Trebuie prealocat.

Creare coadă de comenzi

```
cl_command_queue clCreateCommandQueueWithProperties(
        ↓ cl_context context,
        ↓ cl_device_id device,
        ↓ cl_command_queue_properties properties,
        ↑  cl_int* errcode_ret);
```

Întoarce coada creată.

Creare program (grup de funcții)

```
cl_program clCreateProgramWithSource(↓ cl_context context,
    ↓ cl_uint count,
    ↓ const char** strings,
    ↓ const size_t* lengths,
    ↑  cl_int* errcode_ret);
```

Creare program

```
cl_program clCreateProgramWithSource(↓ cl_context context,
    ↓ cl_uint count,
    ↓ const char** strings,
    ↓ const size_t* lengths,
    ↑  cl_int* errcode_ret);
```

Context valid.

Creare program

```
cl_program clCreateProgramWithSource(↓ cl_context context,
    ↓ cl_uint count,
    ↓ const char** strings,
    ↓ const size_t* lengths,
    ↑  cl_int* errcode_ret);
```

Șiruri de caractere reprezentând coduri.

Creare program

```
cl_program clCreateProgramWithSource(↓ cl_context context,
    ↓ cl_uint count,
    ↓ const char** strings,
    ↓ const size_t* lengths,
    ↑  cl_int* errcode_ret);
```

Mărimile fiecărui șir de caractere.

Creare program

```
cl_program clCreateProgramWithSource(↓ cl_context context,
    ↓ cl_uint count,
    ↓ const char** strings,
    ↓ const size_t* lengths,
    ↑  cl_int* errcode_ret);
```

Numărul șirurilor de caractere.

Creare program

```
cl_program clCreateProgramWithSource(↓ cl_context context,
    ↓ cl_uint count,
    ↓ const char** strings,
    ↓ const size_t* lengths,
    ↑ cl_int* errcode_ret);
```

Întoarce CL_SUCCESS sau cod de eroare. Trebuie prealocat.

Creare program

```
cl_program clCreateProgramWithSource(↓ cl_context context,
    ↓ cl_uint count,
    ↓ const char** strings,
    ↓ const size_t* lengths,
    ↑  cl_int* errcode_ret);
```

Întoarce un program.

Compilare și link-are program

```
cl_int clBuildProgram(cl_program program,
        cl_uint num_devices,
        const cl_device_id* device_list,
        const char* options,
        void (CL_CALLBACK* pfn_notify)(cl_program program,
                void* user_data),
        void* user_data);
```

Compilare și link-are program

```
cl_int clBuildProgram(cl_program program,
        cl_uint num_devices,
        const cl_device_id* device_list,
        const char* options,
        void (CL_CALLBACK* pfn_notify)(cl_program program,
                void* user_data),
        void* user_data);
```

Programul ce va fi compilat și linkat.

Compilare și link-are program

```
cl_int clBuildProgram(cl_program program,
        cl_uint num_devices,
        const cl_device_id* device_list,
        const char* options,
        void (CL_CALLBACK* pfn_notify)(cl_program program,
                void* user_data),
        void* user_data);
```

Dispozitivele pentru care se va face compilarea.

Compilare și link-are program

```
cl_int clBuildProgram(cl_program program,
        cl_uint num_devices,
        const cl_device_id* device_list,
        const char* options,
        void (CL_CALLBACK* pfn_notify)(cl_program program,
                void* user_data),
        void* user_data);
```

Numărul acestor dispozitive.

Pentru noi: 0 – toate dispozitivele din program.

Compilare și link-are program

```
cl_int clBuildProgram(cl_program program,
        cl_uint num_devices,
        const cl_device_id* device_list,
        const char* options,
        void (CL_CALLBACK* pfn_notify)(cl_program program,
                void* user_data),
        void* user_data);
```

Șir de caractere cu opțiuni de compilare.

Pentru noi: NULL.

Compilare și link-are program

```
cl_int clBuildProgram(cl_program program,
        cl_uint num_devices,
        const cl_device_id* device_list,
        const char* options,
        void (CL_CALLBACK* pfn_notify)(cl_program program,
                void* user_data),
        void* user_data);
```

Funcție callback care va fi apelată pentru a raporta erori din context.
Pentru noi: NULL

Compilare și link-are program

```
cl_int clBuildProgram(cl_program program,
        cl_uint num_devices,
        const cl_device_id* device_list,
        const char* options,
        void (CL_CALLBACK* pfn_notify)(cl_program program,
                void* user_data),
        void* user_data);
```

Date cu care va fi apelată funcția de callback.
Pentru noi: NULL.

Compilare și link-are program

```
cl_int clBuildProgram(cl_program program,
        cl_uint num_devices,
        const cl_device_id* device_list,
        const char* options,
        void (CL_CALLBACK* pfn_notify)(cl_program program,
                void* user_data),
        void* user_data);
```

Întoarce CL_SUCCESS sau cod de eroare.

Informații despre compilare (warnings/errors)

```
cl_int clGetProgramBuildInfo(cl_program program,
        cl_device_id device,
        cl_program_build_info param_name,
        size_t param_value_size,
        void* param_value,
        size_t* param_value_size_ret);
```

Informații despre compilare

```
cl_int clGetProgramBuildInfo(cl_program program,
        cl_device_id device,
        cl_program_build_info param_name,
        size_t param_value_size,
        void* param_value,
        size_t* param_value_size_ret);
```

Programul pentru care vrem informații.

Informații despre compilare

```
cl_int clGetProgramBuildInfo(cl_program program,
        cl_device_id device,
        cl_program_build_info param_name,
        size_t param_value_size,
        void* param_value,
        size_t* param_value_size_ret);
```

Dispozitivul pentru care dorim informații (s-ar putea doar pe unele să avem eroare).

Informații despre compilare

```
cl_int clGetProgramBuildInfo(cl_program program,
        cl_device_id device,
        cl_program_build_info param_name,
        size_t param_value_size,
        void* param_value,
        size_t* param_value_size_ret);
```

Ce informații dorim.

Pentru noi: CL_PROGRAM_BUILD_LOG

Informații despre compilare

```
cl_int clGetProgramBuildInfo(cl_program program,
        cl_device_id device,
        cl_program_build_info param_name,
        size_t param_value_size,
        void* param_value,
        size_t* param_value_size_ret);
```

Informațiile sunt stocate aici.

Pentru noi: Un vector de caractere prealocat în care se vor scrie warnings/erors.

Informații despre compilare

```
cl_int clGetProgramBuildInfo(cl_program program,
        cl_device_id device,
        cl_program_build_info param_name,
        size_t param_value_size,
        void* param_value,
        size_t* param_value_size_ret);
```

Mărimea vectorului.

Informații despre compilare

```
cl_int clGetProgramBuildInfo(cl_program program,
        cl_device_id device,
        cl_program_build_info param_name,
        size_t param_value_size,
        void* param_value,
        size_t* param_value_size_ret);
```

Numărul de caractere întors.

Informații despre compilare

```
cl_int clGetProgramBuildInfo(cl_program program,
        cl_device_id device,
        cl_program_build_info param_name,
        size_t param_value_size,
        void* param_value,
        size_t* param_value_size_ret);
```

Întoarce CL_SUCCESS sau cod de eroare.

Creare kernel-ului care va fi rulat

```
cl_kernel clCreateKernel(cl_program program,
        const char* kernel_name,
        cl_int* errcode_ret);
```

Creare kernel-ului care va fi rulat

```
cl_kernel clCreateKernel(cl_program program,
        const char* kernel_name,
        cl_int* errcode_ret);
```

Programul pentru care dorim kernel.

Creare kernel-ului care va fi rulat

```
cl_kernel clCreateKernel(cl_program program,
        const char* kernel_name,
        cl_int* errcode_ret);
```

Numele kernel-ului ca șir de caractere.

Trebuie să existe compilat în program și să aibă în față **__kernel**

Creare kernel-ului care va fi rulat

```
cl_kernel clCreateKernel(cl_program program,
        const char* kernel_name,
        cl_int* errcode_ret);
```

Întoarce CL_SUCCESS sau cod de eroare. Trebuie prealocat.

Creare kernel-ului care va fi rulat

```
cl_kernel clCreateKernel(cl_program program,
        const char* kernel_name,
        cl_int* errcode_ret);
```

Întoarce o referință către kernelul-ul gata de rulat.

Setare argument apelare kernel-uri

```
cl_int clSetKernelArg(cl_kernel kernel,
        cl_uint arg_index,
        size_t arg_size,
        const void* arg_value);
```

Setare argument apelare kernel-uri

```
cl_int clSetKernelArg(cl_kernel kernel,
        cl_uint arg_index,
        size_t arg_size,
        const void* arg_value);
```

Kernel-ul pentru care se setează argumentul.

Setare argument apelare kernel-uri

```
cl_int clSetKernelArg(cl_kernel kernel,
        cl_uint arg_index,
        size_t arg_size,
        const void* arg_value);
```

Al câtelea argument setăm.

Setare argument apelare kernel-uri

```
cl_int clSetKernelArg(cl_kernel kernel,
        cl_uint arg_index,
        size_t arg_size,
        const void* arg_value);
```

Datele care vor fi transmise argumentului.

Setare argument apelare kernel-uri

```
cl_int clSetKernelArg(cl_kernel kernel,
        cl_uint arg_index,
        size_t arg_size,
        const void* arg_value);
```

Mărimea datelor argumentului setat.

Setare argument apelare kernel-uri

```
cl_int clSetKernelArg(cl_kernel kernel,
        cl_uint arg_index,
        size_t arg_size,
        const void* arg_value);
```

Întoarce CL_SUCCESS sau cod de eroare.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Coada pe care se rulează kernelul.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Kernel-ul care va fi rulat.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Numărul de dimensiuni în care se vor rula kernel-urile (0, CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS).

În general 1, 2 sau 3.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Un vector de work_dim elemente, fiecare element reprezentând offsetul pe acea dimensiune. Poate să fie NULL.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Un vector de work_dim elemente, fiecare element reprezentând mărimea pe acea dimensiune.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Un vector de work_dim elemente, fiecare element reprezentând mărimea pe acea dimensiune a unui work-group.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Evenimente care să se termine înainte să înceapă kernel-urile.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Numărul de evenimente.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Eveniment ce va fi activat după ce au terminat de executat kernel-urile.

Executarea unui kernel pe device(uri)

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
        cl_kernel kernel,
        cl_uint work_dim,
        const size_t* global_work_offset,
        const size_t* global_work_size,
        const size_t* local_work_size,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Întoarce CL_SUCCESS sau cod de eroare.

Alocare memorie pe device

```
cl_mem clCreateBuffer(cl_context context,
        cl_mem_flags flags,
        size_t size,
        void* host_ptr,
        cl_int* errcode_ret);
```

Alocare memorie pe device

```
cl_mem clCreateBuffer(cl_context context,
        cl_mem_flags flags,
        size_t size,
        void* host_ptr,
        cl_int* errcode_ret);
```

Contextul pentru care se alocă memorie.

Alocare memorie pe device

```
cl_mem clCreateBuffer(cl_context context,
        cl_mem_flags flags,
        size_t size,
        void* host_ptr,
        cl_int* errcode_ret);
```

Flag-uri pentru alocarea de memorie.

Ex: CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY, CL_MEM_READ_ONLY

Alocare memorie pe device

```
cl_mem clCreateBuffer(cl_context context,
        cl_mem_flags flags,
        size_t size,
        void* host_ptr,
        cl_int* errcode_ret);
```

Numărul de bytes care să fie alocat.

Alocare memorie pe device

```
cl_mem clCreateBuffer(cl_context context,
        cl_mem_flags flags,
        size_t size,
        void* host_ptr,
        cl_int* errcode_ret);
```

Pointer către memorie host. Posibilitate copiere.

Pentru noi: NULL.

Alocare memorie pe device

```
cl_mem clCreateBuffer(cl_context context,
        cl_mem_flags flags,
        size_t size,
        void* host_ptr,
        cl_int* errcode_ret);
```

Întoarce CL_SUCCESS sau cod de eroare. Trebuie prealocat.

Alocare memorie pe device

```
cl_mem clCreateBuffer(cl_context context,
        cl_mem_flags flags,
        size_t size,
        void* host_ptr,
        cl_int* errcode_ret);
```

Pointer către zona de memorie de pe device.

Memoria nu poate fi accesată de pe host decât cu funcții speciale (read/write/copy).

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Coada pe care se va trimite această comandă.

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Pointer către memoria de pe device.

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Dacă operația va fi blocantă (CL_TRUE) sau neblocantă (CL_FALSE).

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Offset în memoria device.

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Numărul de bytes care vor fi mutați.

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Pointer către zona de memorie host.

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Evenimente care trebuie să se termine înainte acestei operații.

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Numărul de evenimente.

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Eveniment ce se va activa în momentul în care operația se termină.

Copiere memorie din host pe device

```
cl_int clEnqueueWriteBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_write,
        size_t offset,
        size_t size,
        const void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Întoarce CL_SUCCESS sau cod de eroare.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Coada pe care se va trimite această comandă.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Pointer către memoria de pe device.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Dacă operația va fi blocantă (CL_TRUE) sau neblocantă (CL_FALSE).

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Offset în memoria device.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Numărul de bytes care vor fi mutați.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Pointer către zona de memorie host.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Evenimente care trebuie să se termine înainte acestei operații.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Numărul de evenimente.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Eveniment ce se va activa în momentul în care operația se termină.

Copiere memorie de pe device pe host

```
cl_int clEnqueueReadBuffer(cl_command_queue command_queue,
        cl_mem buffer,
        cl_bool blocking_read,
        size_t offset,
        size_t size,
        void* ptr,
        cl_uint num_events_in_wait_list,
        const cl_event* event_wait_list,
        cl_event* event);
```

Întoarce CL_SUCCESS sau cod de eroare.

Așteaptă terminarea operațiilor din coadă

```
cl_int clFinish(
        cl_command_queue command_queue);
```

Așteaptă terminarea operațiilor din coadă

```
cl_int clFinish(
        cl_command_queue command_queue);
```

Coada de comenzi.

Așteaptă terminarea operațiilor din coadă

```
cl_int clFinish(
        cl_command_queue command_queue);
```

Întoarce CL_SUCCESS sau cod de eroare.

Dealocare resurse

```
cl_int clReleaseMemObject(cl_mem memobj)
cl_int clReleaseProgram(cl_program program)
cl_int clReleaseKernel(cl_kernel kernel)
cl_int clReleaseCommandQueue(cl_command_queue command_queue)
cl_int clReleaseContext(cl_context context)
```

OpenCL C Language

Address Space Qualifiers

- __global
  - Accesibil de toate work-item-urile.
- __local
  - Accesibil doar de work-item-urile unui work-group.
- __private
  - Accesibil de un singur work-item.
- __constant

Built in Functions

- uint **get_work_dim**()
  - Cu câte dimensiuni a fost rulat kernel-ul.
- size_t **get_global_size**(uint *dimindx*)
  - Mărimea globală pentru dimensiunea dimindx.
- size_t **get_global_id**(uint *dimindx*)
  - Locația în dimensiunea dimindx.
- size_t **get_local_size**(uint *dimindx*)
  - Mărimea work-group-ului pe dimensiunea dimindx.
- size_t **get_local_id**(uint *dimindx*)
  - Locația în work-group pe dimensiunea dimindx.

Synchronization

- void **barrier**(cl_mem_fence_flags *flags*)
- void **work_group_barrier**(cl_mem_fence_flags *flags*)
  - Toate work-item-urile unui work-group trebuie să aștepte la barieră pentru a trece mai departe.
  - Dacă bariera în if toate să intre pe aceeași ramură a if-ului.
  - Dacă bariera în for toate să facă același număr de iterații ale for-ului.

Synchronization

- No mutex
- No semaphore
- Yes atomics... Tons of atomics.
  - atomic_add
  - atomic_sub
  - atomic_xchg
  - atomic_inc
  - atomic_min
  - atomic_and