OpenCL

Johannes Hackel Tim Illner Marcel Wesberg

25. Oktober 2012

Gliederung

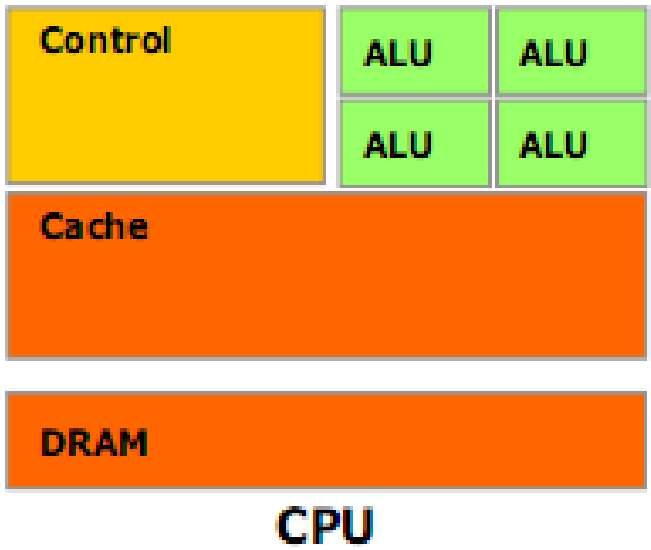
Was ist OpenCL?

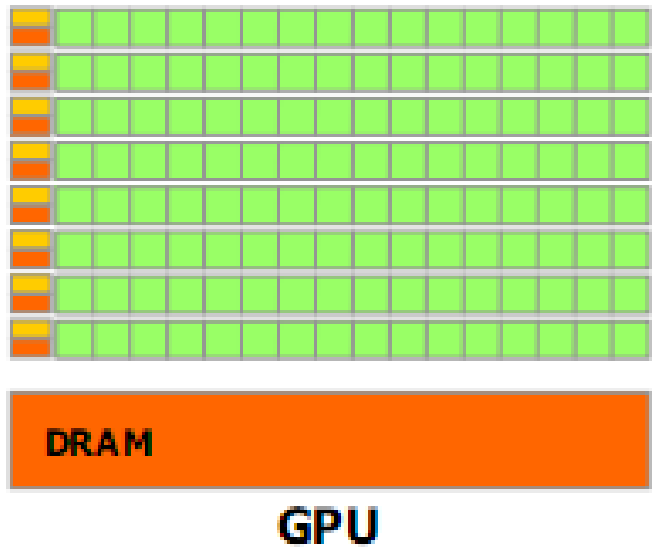
Geschichte

Unterstütze Geräte/Treiber

Aufbau GPUs

- Grafikprozessor besteht aus Recheneinheiten, ähnlich CPU:
- Ausführung Operationen der Arithmetik, Decodierung und Datentransfer
- Verarbeitung paralleler Prozesse (zB. mathematischer Operationen) bei konstanter Taktrate
- Durch breiten Datenbus mit Speichereinheit verbunden, großer Datendurchsatz
- Verarbeitung vieler Daten gleichzeitig:
- graphische, dreidimensionale Objekte, Beleuchtung, Farben und Texturen
- Auswertung wissenschaftliche Daten
- Simulation komplexer physikalischer Systeme (zB. viele kleine Teilchen)


Control
ALU
ALU
ALU
ALU
Cache
DRAM
CPU


DRAM
GPU

Die OpenCL

Kernel

```
float Sum(float x, float y)
{
 return(x + y);
}

__kernel void Calculate(__global float* input,
 __global float* output)
{

}
```

einfache Datentypen

- bool: Boolescher Wahrheitswert, true oder false.
- char: Vorzeichenbehaftete 8-Bit Ganzzahl.
- uchar, unsigned char: Vorzeichenlose 8-Bit Ganzzahl.
- short: Vorzeichenbehaftete 16-Bit Ganzzahl.
- ushort, unsigned short: Vorzeichenlose 16-Bit Ganzzahl.
- int: Vorzeichenbehaftete 32-Bit Ganzzahl.
- uint, unsigned int: Vorzeichenlose 32-Bit Ganzzahl.
- long: Vorzeichenbehaftete 64-Bit Ganzzahl.
- ulong, unsigned long: Vorzeichenlose 64-Bit Ganzzahl.
- float: 32-Bit Gleitkommazahl.
- half: 16-Bit Gleitkommazahl.

Vektordatentypen

- bestehend aus 2, 3, 4, 8 oder 16 Elementen
- vom Typ char, uchar, short, ushort, int, uint, long, ulong oder float
- verschiedene Möglichkeiten der Deklaration und Initialisierung

```
float4 a = (float4)(0.0, 1.0, 2.0, 3.0);
float2 b = (float2)(1.0, 2.0);
float2 c = (float2)(0.0);
float4 d = (float4)(b, c);
float8 e = (float8)(b, a, c);
```

Vektordatentypen

- Komponentenweise Addition, Subtraktion und Multiplikation:

```
float4 a, b, c;
int8 d, e, f, g;
c = a + b;
c = a - b;
f = d * e;
g = 2 * f;
```

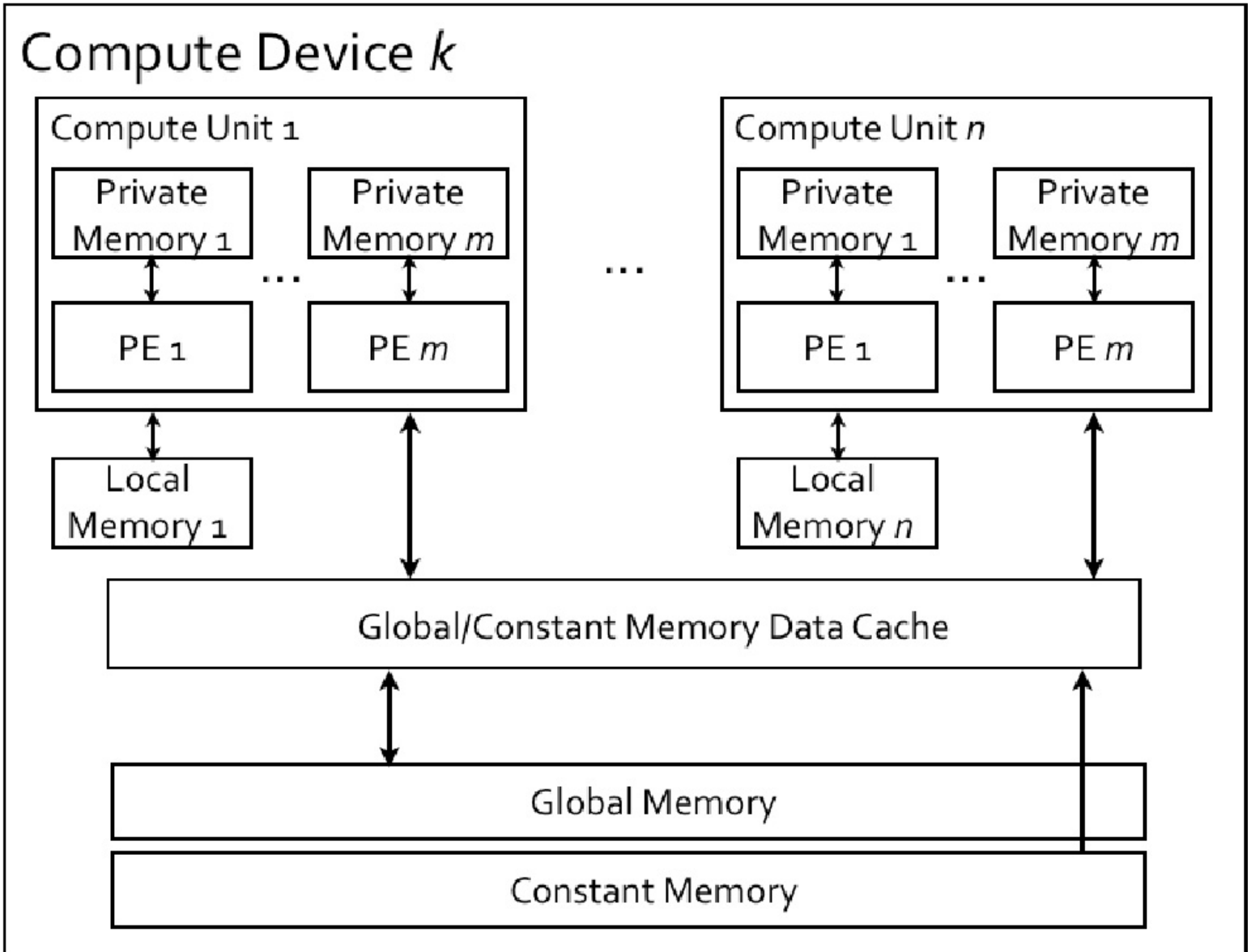
Vektordatentypen

Zugriff auf einzelne Komponenten:

- bei 2, 3 oder 4 Elementen durch x, y, z, w
- bei 8 Elementen: s0 bis s7
- bei 16 Elementen: s0 bis s9 und sa bis sf

```
float4 a;
float b = a.z;
float c = a.s2;
float16 d;
float e = d.sf;
float2 f, g;
float8 h;
float16 i;
g.xy = f.yx;
g.s01 = f.s10;
h.s01234567 = i.sfedcba98;
```

Speicherbereiche



Speicherbereiche

privater Speicher:

- __private
- Variablen die in einer Funktion deklariert wurden und Funktionsargumente
- nur in dieser Funktion zugänglich
- existieren nur für die jeweilige Kernel-Instanz

lokaler Speicher:

- __local
- werden von allen Kernel-Instanzen in einer Arbeitsgruppe gemeinsam genutzt
- jede Arbeitsgruppe besitzt eigene Kopie

Speicherbereiche

globaler Speicher:

- __global
- Zugriff von Host und Client möglich
- üblicherweise ein Zeiger auf Speicherbereich
- alle Kernel-Instanzen greifen auf die selben Daten zu

Konstantenspeicher:

- __constant
- nur lesbar
- kann in lokalen Speicher liegen

Funktionen für die Arbeitsverwaltung

- uint get_work_dim(void)
- size_t get_global_size(uint dim)
- size_t get_global_id(uint dim)
- size_t get_local_size(uint dim)
- size_t get_local_id(uint dim)
- size_t get_num_groups(uint dim)
- size_t get_group_id(uint dim)
- size_t get_global_offset(uint dim)

Mathematische Funktionen

- Exponentialfunktion und Logarithmus: exp, exp2, exp10, log, log2, log10
- Trigonometrische Funktionen: sin, cos, tan, asin, acos, atan, atan2
- Hyperbolische Funktionen: sinh, cosh, tanh, asinh, acosh, atanh
- Wurzeln: sqrt, cbrt
- Potenzen: pow
- Rundungsfunktionen: round, floor, ceil
- weiter Funktionen findet man in der OpenCL-Spezifikation

Die Laufzeitbibliotheken

Khronos OpenCL API Registry

The OpenCL API registry contains specifications of the core API and a portable intermediate representation of OpenCL programs; specifications of Khronos- and vendor-approved OpenCL extensions; header files corresponding to the specifications; and other related documentation.

OpenCL Core API and SPIR Specification, Headers, and Documentation

The current version of OpenCL is OpenCL 1.2.

- OpenCL 1.2 Specification (revision 15, released November 15, 2011).
- OpenCL SPIR (Standard Portable Intermediate Representation) (version 1.0 provisional).
- OpenCL 1.2 Extensions Specification (revision 15, released November 15, 2011).
- OpenCL 1.2 Online Manual Pages.
- OpenCL 1.2 Reference Card (revision 1111, released November 15, 2011).
- All of the following headers should be present in a directory `CL/` (or `OpenCL/` on MacOS X). The single header file opencl.h includes other headers as appropriate for the target platform, and simply including `opencl.h` should be all that most applications need to do.
  - opencl.h - OpenCL 1.2 Single Header File for Applications.
  - cl_platform.h - OpenCL 1.2 Platform-Dependent Macros.
  - cl.h - OpenCL 1.2 Core API Header File.
  - cl_ext.h - OpenCL 1.2 Extensions Header File.
  - cl_dx9_media_sharing.h - OpenCL 1.2 Khronos OpenCL/Direct3D 9 Media Sharing Extensions Header File.
  - cl_d3d10.h - OpenCL 1.2 Khronos OpenCL/Direct3D 10 Extensions Header File.
  - cl_d3d11.h - OpenCL 1.2 Khronos OpenCL/Direct3D 11 Extensions Header File.
  - cl_gl.h - OpenCL 1.2 Khronos OpenCL/OpenGL Extensions Header File.
  - cl_gl_ext.h - OpenCL 1.2 Vendor OpenCL/OpenGL Extensions Header File.
  - cl.hpp - OpenCL 1.1 C++ Bindings Header File, implementing the C++ Bindings Specification. **This header works for all versions of OpenCL, but has not yet been updated with new OpenCL 1.2 entry points.**
  - Extension template for writing an OpenCL extension specification. Extensions in the registry (listed below) follow the structure of this document, which describes the purpose of each section in an extension specification.

Eine Übersicht der verfügbaren Laufzeitbibliotheken erhält man auf `http://www.khronos.org/registry/cl/`

Plattformen

```
clGetPlatformIDs(cl_uint          /* num_entries */,
 cl_platform_id * /* platforms */,
 cl_uint *        /* num_platforms */);
```

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

Geräte

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

```
clCreateSubDevices(cl_device_id                    /* in_device */,
 const cl_device_partition_property* /* properties*/,
 cl_uint                 /* num_devices */,
 cl_device_id *          /* out_devices */,
 cl_uint *               /* num_devices_ret */);
```

Kontexte

```
clGetContextInfo(cl_context      /* context */,
 cl_context_info      /* param_name */,
 size_t               /* param_value_size */,
 void *               /* param_value */,
 size_t *             /* param_value_size_ret */);
```

Speicherverwaltung

```
clGetMemObjectInfo(cl_mem          /* memobj */,
 cl_mem_info       /* param_name */,
 size_t            /* param_value_size */,
 void *            /* param_value */,
 size_t *          /* param_value_size_ret */);

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 */);
```

Programme

```
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 */);
```

Programme

```
clCompileProgram(cl_program           /* program */,
 cl_uint                  /* num_devices */,
 const cl_device_id *     /* device_list */,
 const char *             /* options */,
 cl_uint                  /* num_input_headers */,
 const cl_program *       /* input_headers */,
 const char **            /* header_include_names */,
 void (CL_CALLBACK *      /* pfn_notify */)
 (cl_program /* program */,void * /* user_data */),
 void *                   /* user_data */);
```

Kernel

```
clCreateKernelsInProgram(cl_program  /* program */,
                         cl_uint     /* num_kernels */,
                         cl_kernel * /* kernels */,
                         cl_uint *   /* num_kernels_ret */);
```

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

Warteschlangen

```
clCreateCommandQueue(cl_context                     /* context */,
                     cl_device_id                   /* device */,
    cl_command_queue_properties                     /* properties */,
                     cl_int *                       /* errcode_ret */);
```

```
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 */);
```

Ereignisse

```
clWaitForEvents(cl_uint   /* num_events */,
 const cl_event *     /* event_list */);


clGetEventInfo(cl_event           /* event */,
 cl_event_info      /* param_name */,
 size_t             /* param_value_size */,
 void *             /* param_value */,
 size_t *           /* param_value_size_ret */);


clCreateUserEvent(cl_context     /* context */,
 cl_int *       /* errcode_ret */)
```

Marker und Barrieren

```
clEnqueueMarker(
    cl_command_queue    /* command_queue */,
    cl_event *          /* event */);
```

```
clEnqueueBarrier(
    cl_command_queue /* command_queue */)
```

Kompilierung und Ausführung

Quellen

- http://www.khronos.org/
- hexagon.fi.tartu.ee/~manuel/teaching/gpu.pdf
- http://www.zdnet.de/wp-content/uploads/legacy_images/news/201004/aws-gpu-v6.png
- http://developer.amd.com/Resources/documentation/articles/PublishingImages/opencl_figure5.jpg