OpenCL

Torsten Kammer

Inhalt

- Was ist OpenCL und wofür braucht man das?
- Kontext erzeugen
- Speicher
- Code schreiben
- Mit OpenGL zusammen

Was ist OpenCL?

- Framework für GPGPU-Programmierung
- Industriestandard analog zu OpenGL
- Erfunden von Apple
- Erstmals verfügbar mit 10.6 Snow Leopard
- Mit 10.6.8/10.7 deutliche Verbesserung

Wieso?

	CPU	Grafikkarte
Typ	Core i7 870	Radeon HD 5750

(iMac 27” Mitte 2010)

Wieso?

	CPU	Grafikkarte
Typ	Core i7 870	Radeon HD 5750
Takt	2,93–3,06 Ghz	0,7 Ghz

(iMac 27” Mitte 2010)

Wieso?

	CPU	Grafikkarte
Typ	Core i7 870	Radeon HD 5750
Takt	2,93–3,06 Ghz	0,7 Ghz
Cores	4 (8 mit HT)	720

(iMac 27” Mitte 2010)

Wieso?

	CPU	Grafikkarte
Typ	Core i7 870	Radeon HD 5750
Takt	2,93–3,06 Ghz	0,7 Ghz
Cores	4 (8 mit HT)	720
Leistung	bis 51,2 GFLOPS	1008

(iMac 27” Mitte 2010)

Was OpenCL nicht ist

Für jedes Programm geeignet

Automatisch aktivierbar

Verbunden mit Grand Central Dispatch

Wofür OpenCL gut ist

- Parallel - viele kleine Daten gleichzeitig bearbeiten
- Große Mathematische Probleme, Simulationen und ähnliches
- Integration mit OpenGL
  - z.B. Vertex-Buffer bearbeiten

API Design

- Wie OpenGL, nur…

Objekte durch ints benannt	cl_mem, cl_kernel, cl_…
glBind…	Explizit als Parameter
glGetError()	Rückgabewert od. Callback
glDelete…	clRetain…, clRelease…
GLint	cl_int

clGetDeviceIDs(…)

clCreateContext(…)

clCreateCommandQueue(…)

CGLContextObj

CGLGetShareGroup(…)

CGLShareGroupObj

clCreateContext(…)

cl_context

clGetContextInfo(…)

cl_device_id

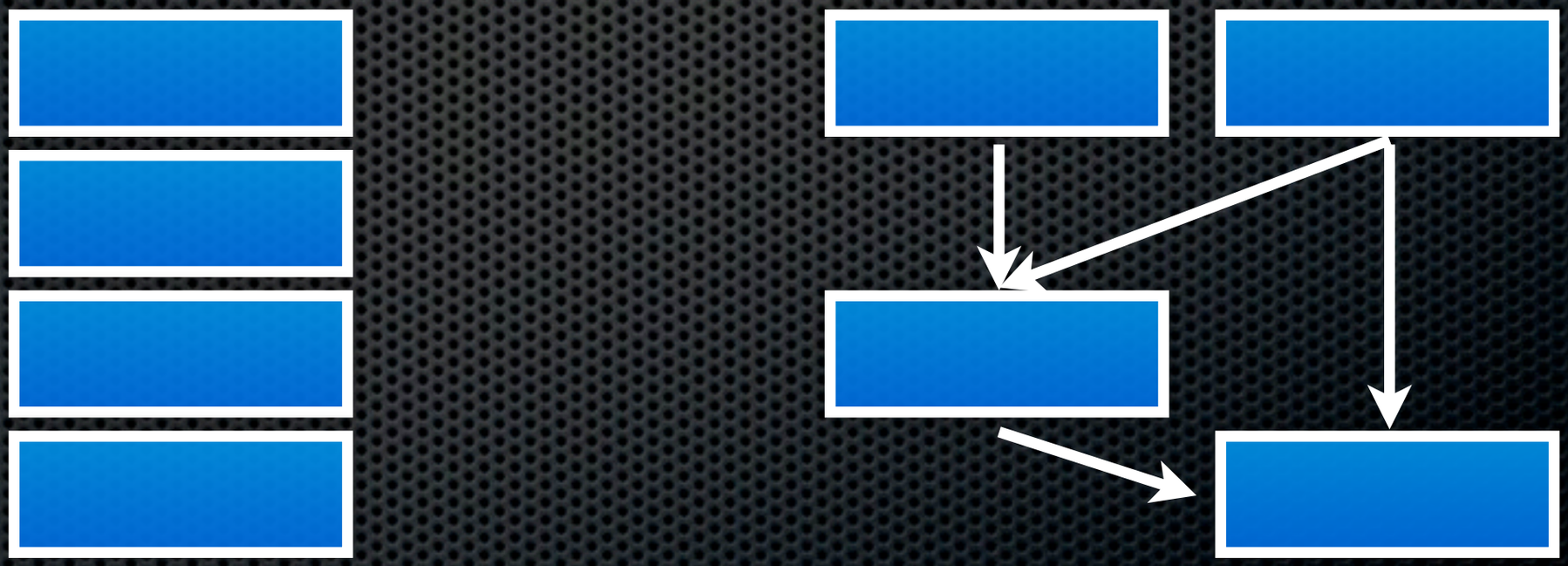
clCreateCommandQueue(…)

cl_command_queue

Demo

Command Queue

- Organisiert alle Operationen für ein Device
- Kann rein linear oder out of order sein (nicht überall unterstützt)



Events

- Synchronisierung für Out-of-order Queues

Events, die zuerst fertig sein müssen

```
clEnqueue…(…, cl_uint, const cl_event *, cl_event *)
```

Rückgabe: Event für diese Operation
(muss released werden)

- Beenden von Operationen

```
clWaitForEvents(cl_uint, const cl_event *)
```

Daten

- Daten müssen auf Device kopiert werden
- Gespeichert in Buffer oder Image
- Kopieren bei Erstellen oder über Queue
  - Wahlweise blockierend oder asynchron

Demo

Echter Code

- Code wird in Kernels gespeichert (entspricht ungefähr Shader oder Kernel in CoreImage)
- Programmiert in C (mit Erweiterungen)
- Gespeichert als Quellcode

Programmieren

- Wie C, aber…
  - float2, float3, float4 u.ä. Vektortypen
  - Keine Standardbibliothek, (fast) nur Mathe
- Nicht wie GLSL!
  - Keine Matrix-Typen
  - Syntax für Bilder ganz anders

Ausführen

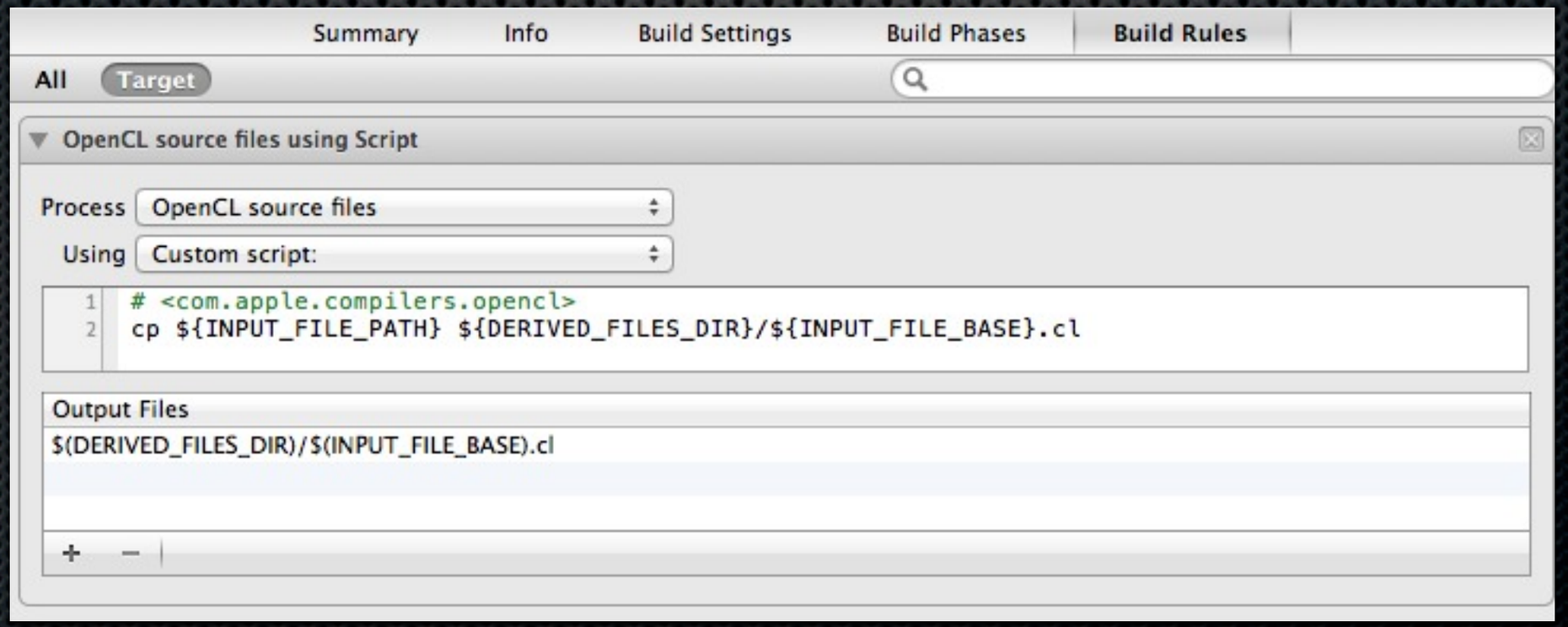
- Code wird in Workgroups ausgeführt
- Kommunikation nur in selber Workgroup
- Workgroup-Größe begrenzt
- Threads teilen Instruction Pointer

Speicher

- Threads sind in Workgroups zusammengefasst
- Alle Pointer müssen Ebene zugewiesen kriegen:
  - __global, __constant: Alle Threads teilen sich den Speicher, wird von CPU gesetzt
  - __local: CPU gibt nur Größe vor. Geteilt innerhalb einer Workgroup

Dateien in Xcode

- Syntaxhighlighting (Dateiendung .cl, .opencl)
- Hat auch Compiler, aber unklar was der tut



Demo

OpenGL

- OpenGL und OpenCL können Buffer, Images teilen
  - Vorraussetzung: Kontext mit ShareGroup erstellt
- `clCreateBufferFromGLBuffer()`
- Keine automatische Synchronisierung

Synchronisierung

Verwenden in OpenGL

```
glFinish()
```

```
clEnqueueAcquireGLBuffer(…)
```

Verwenden in OpenCL

```
clEnqueueReleaseGLBuffer(…, &event)
clWaitForEvents(1, &event)
```

Verwenden in OpenGL

Bessere Synchronisierung

Klassisches OpenGL

Verwenden in OpenGL

`glSetFenceAPPLE()` | `fence=glFenceSync()`

Anderes OpenGL

`glFinishFenceAPPLE()` | `glClientWaitSync()`

`clEnqueueAcquireGLBuffer(…)`

Verwenden in OpenCL

```
clEnqueueReleaseGLBuffer(…, &event)
clWaitForEvents(1, &event)
```

Verwenden in OpenGL

Core Profile