Raytracing in GA mittels OpenACC

TECHNISCHE UNIVERSITÄT DARMSTADT

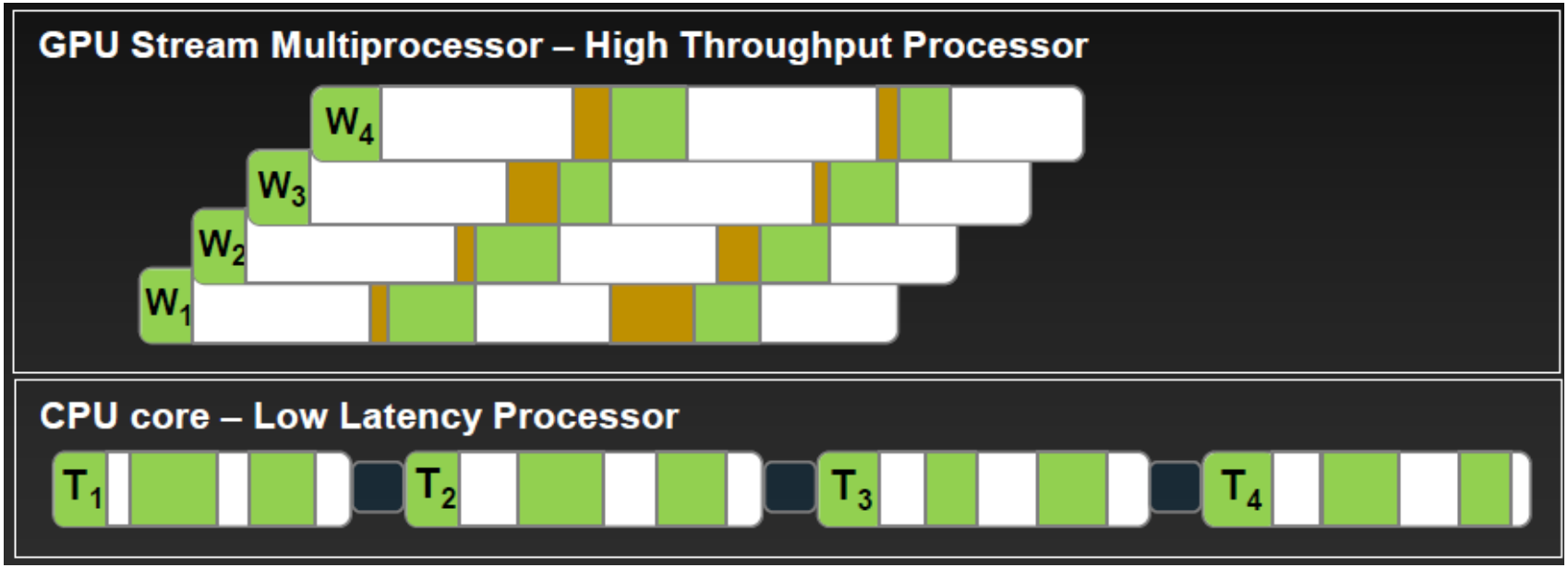
Michael Burger, M.Sc.
FG Scientific Computing
TU Darmstadt
michael.burger@sc.tu-darmstadt.de

Agenda

TECHNISCHE UNIVERSITÄT DARMSTADT

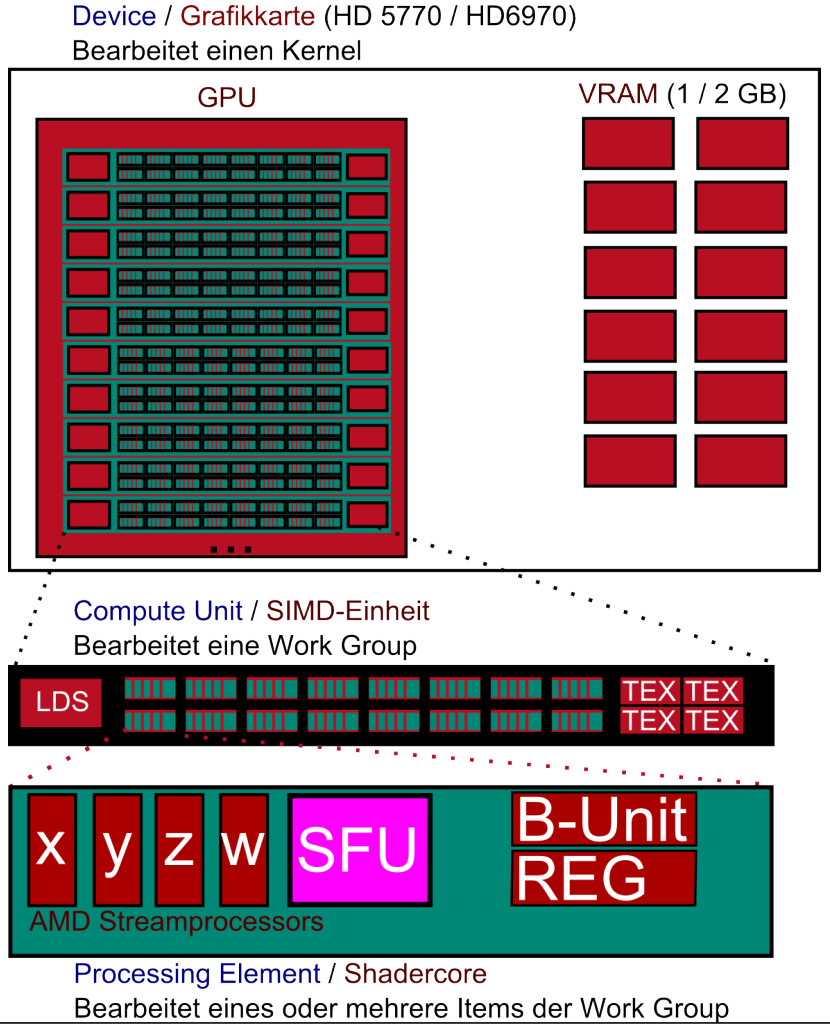
- GPU-Computing und moderne GPU Architekturen
- Raytracing in Geometrischer Algebra
- Resultate des Raytracings auf parallelen Architekturen
- Kurzeinführung OpenACC

Ideen des GPU-Computing



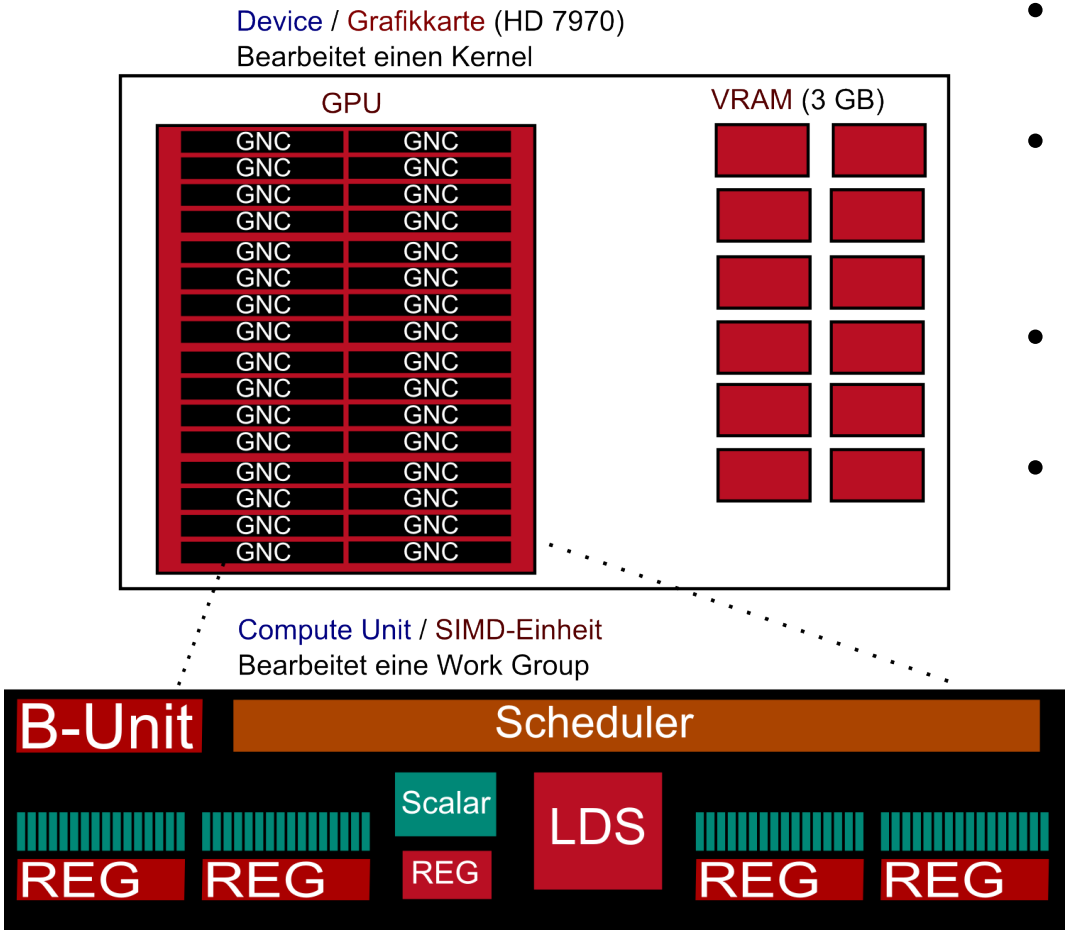
- SIMD, pipelined, seriell
- Viele Berechnungen relativ langsam, gepipelined ausführen statt schnell und seriell
- Gilt auch für Multicore-CPUs

GPU Architektur: AMD Evergreen / Northern Islands



- OpenCL-Hierarchie: Kernel → Groups → Items
- Parallelität der Work Items ausnutzen
- Parallelität der Multivektoren-Berechnungen ausnutzen
- AMDs Verwendung von VLIW → Compilerabhängigkeit
- AMD APP Profiler ermöglicht Analysen

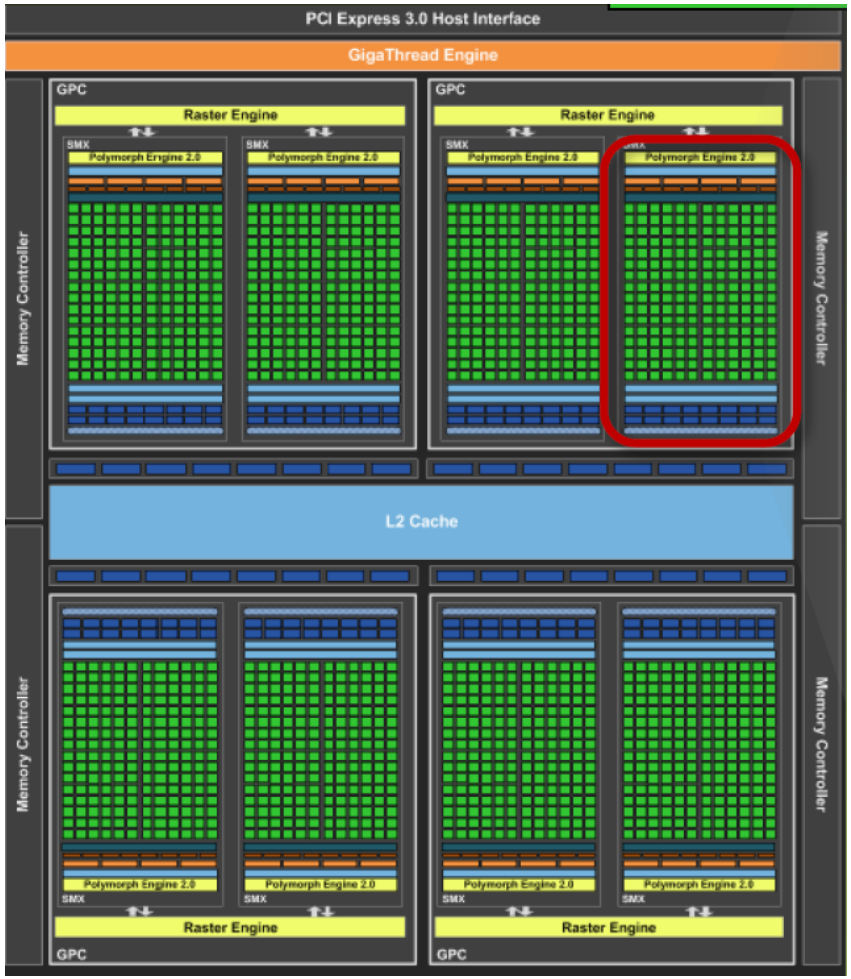
GPU Architektur: Southern Islands

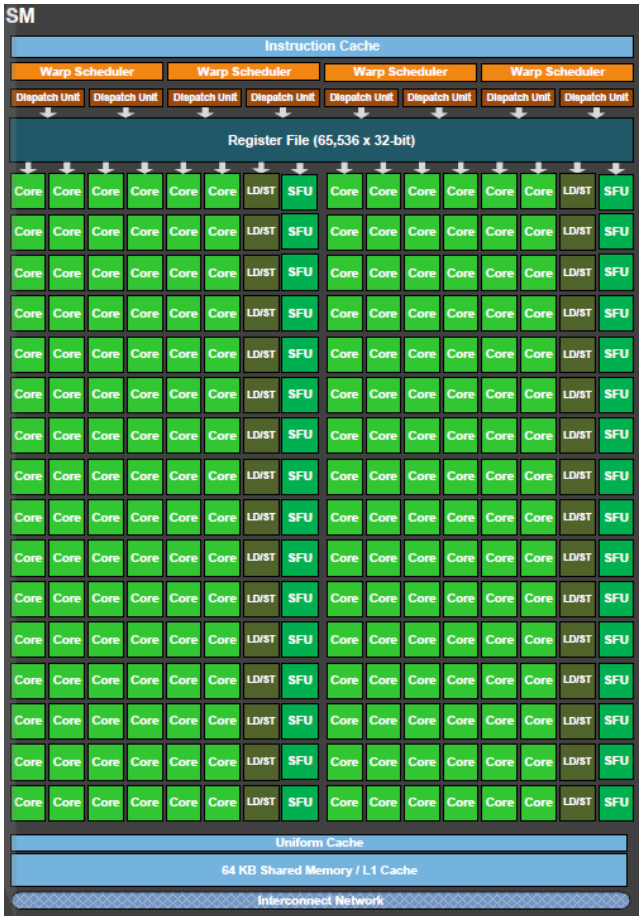


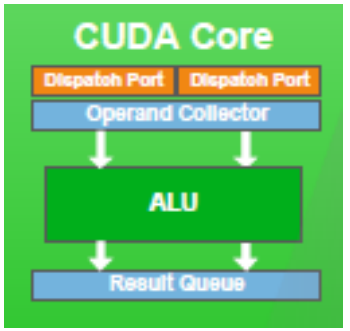
- 4D / 5D zu 1D
- GNC entspricht SIMD (64 Multiplay-ADD)
- Ziel: höhere Auslastung
- Dynamische Ressourcenzuteilung

GPU Architektur: Nvidia Kepler

TECHNISCHE UNIVERSITÄT DARMSTADT

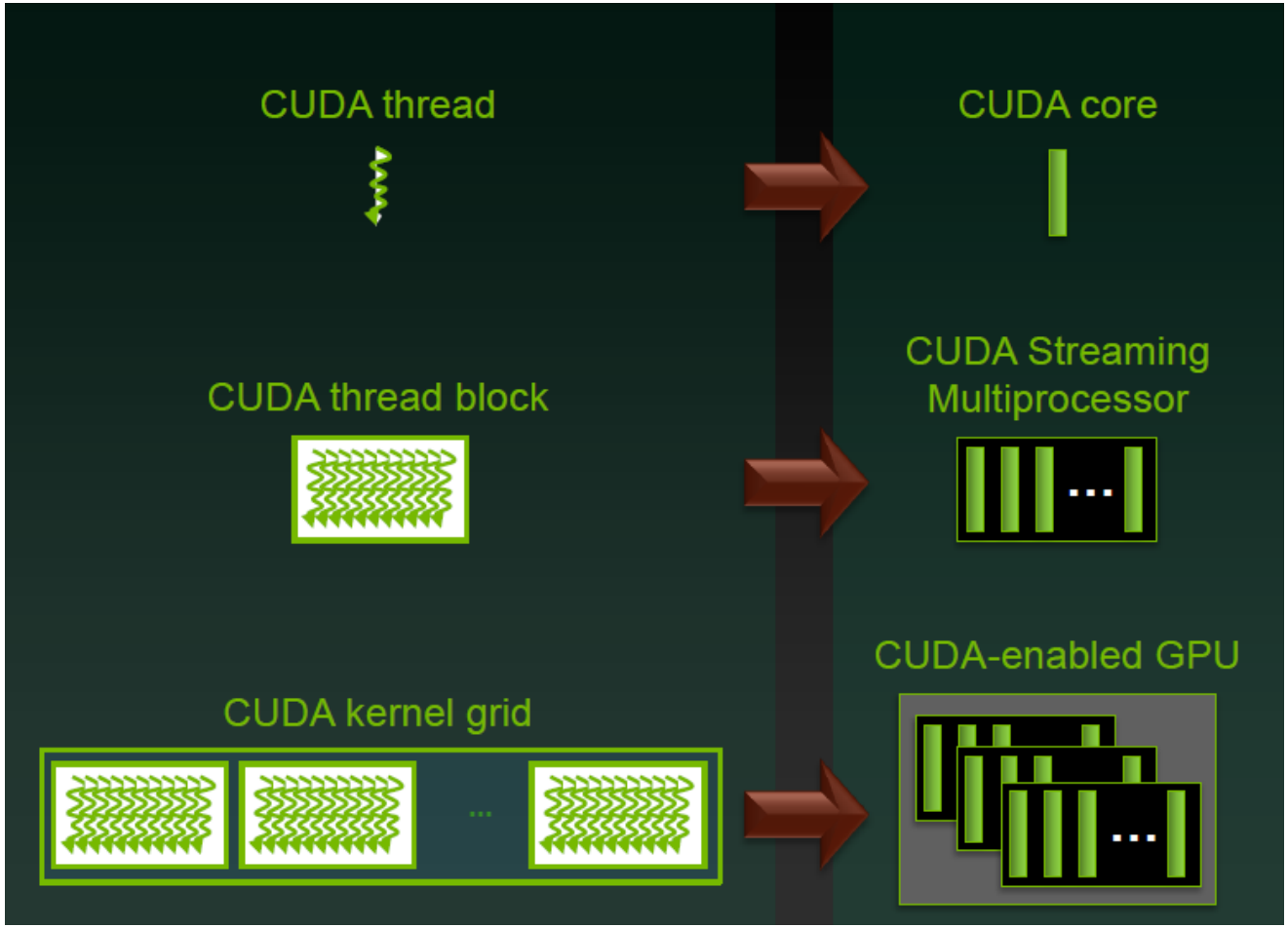




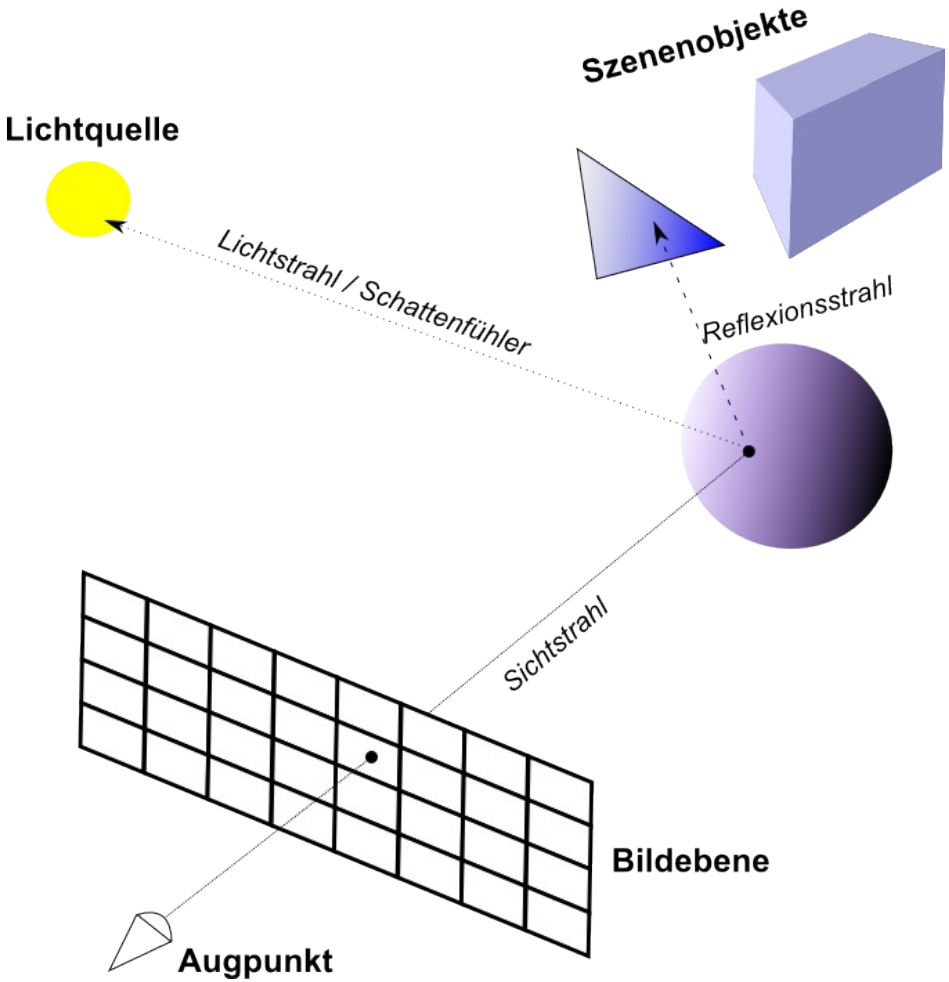


SCIENTIFIC COMPUTING

GPU Architektur



Raytracing



1. Berechne Sichtstrahl

2. Prüfe auf Schnitt mit Objekten

3. Finde den nächsten Schnittpunkt

4. Erzeuge Reflexionsstrahlen und Schattenfühler

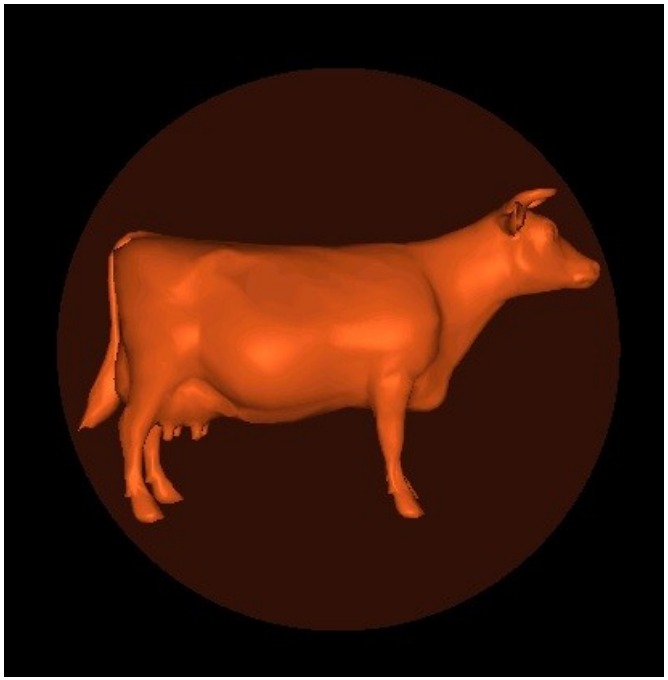
5. Berechne lokale Farbe

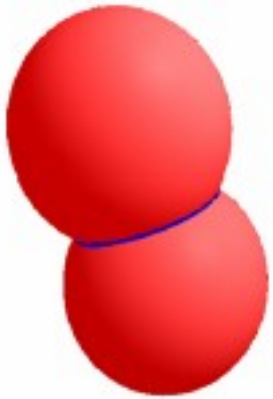
6. Wiederhole mit Reflexionsstrahlen 2-5

Raytracing auf CPU / GPU

TECHNISCHE UNIVERSITÄT DARMSTADT

- Raytracer mit Kugeln und Dreiecken als Szenenobjekte
- Kugelraytracer als Proof of Concept
  → Ergebnisse vergleichbar mit Dreiecks-Raytracern
- Kugeln als Hüllkugeln bei Dreiecken
  → Model-Splitting





$$\text{Kugel } S_1$$
$$\text{Kreis} = S_1 \wedge S_2$$
$$\text{Kugel } S_2$$

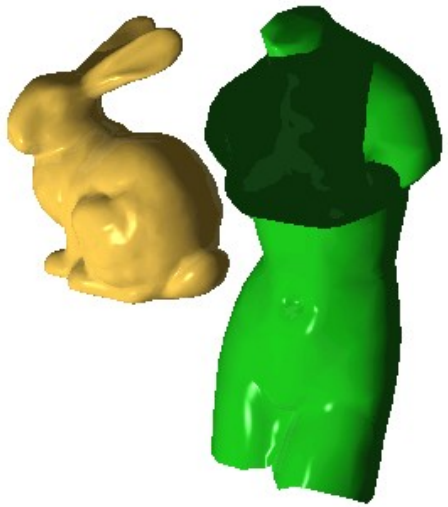
SCIENTIFIC COMPUTING

Raytracing auf CPU / GPU

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mehr Objekte direkt darstellbar (Punkte, Kugeln, Ebenen, …)
- Produkte der Algebra nützlich
- Parallelität durch Multivektoren (pro Blade ein Koeffizient zu berechnen)
- x und n 3D Vektoren mit Basis e_1, e_2 und e_3

Punkt	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$
Kugel	$S = P - \frac{1}{2}r^2 e_\infty$
Ebene	$\pi = \mathbf{n} + d e_\infty$
Kreis	$Z = S_1 \wedge S_2$
Gerade	$L = \pi_1 \wedge \pi_2$
Punktpaar	$Pp = S_1 \wedge S_2 \wedge S_3$

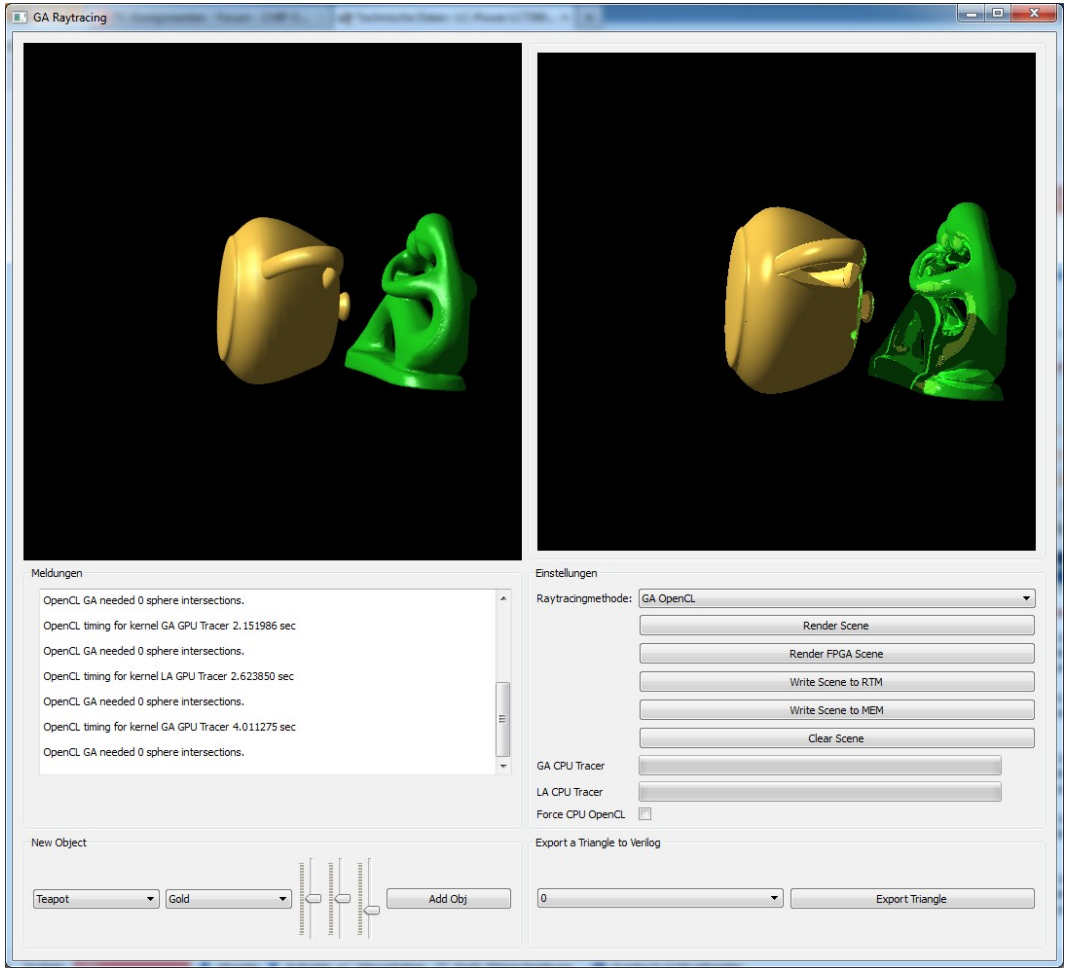


Raytracing auf CPU / GPU

TECHNISCHE UNIVERSITÄT DARMSTADT

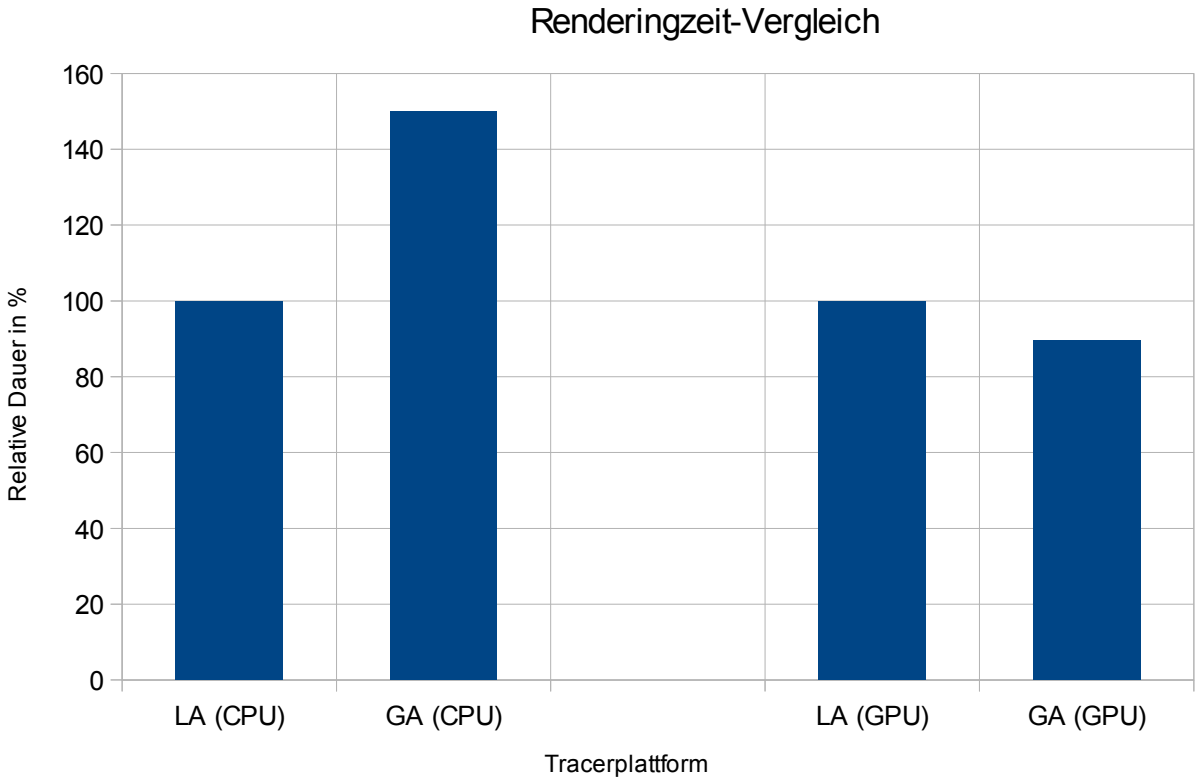
Typ	Sprache	Algebra
Standard CPU	C++	GA / LA
CPU Parallel	C++, OpenMP	GA / LA
Standard GPU	OpenCL	GA / LA
GPU Register Optimized	OpenCL	GA
GPU RAM Optimized	OpenCL	GA / LA
GPU Bounding-Hierarchy	OpenCL	GA / LA

Raytracing auf CPU / GPU



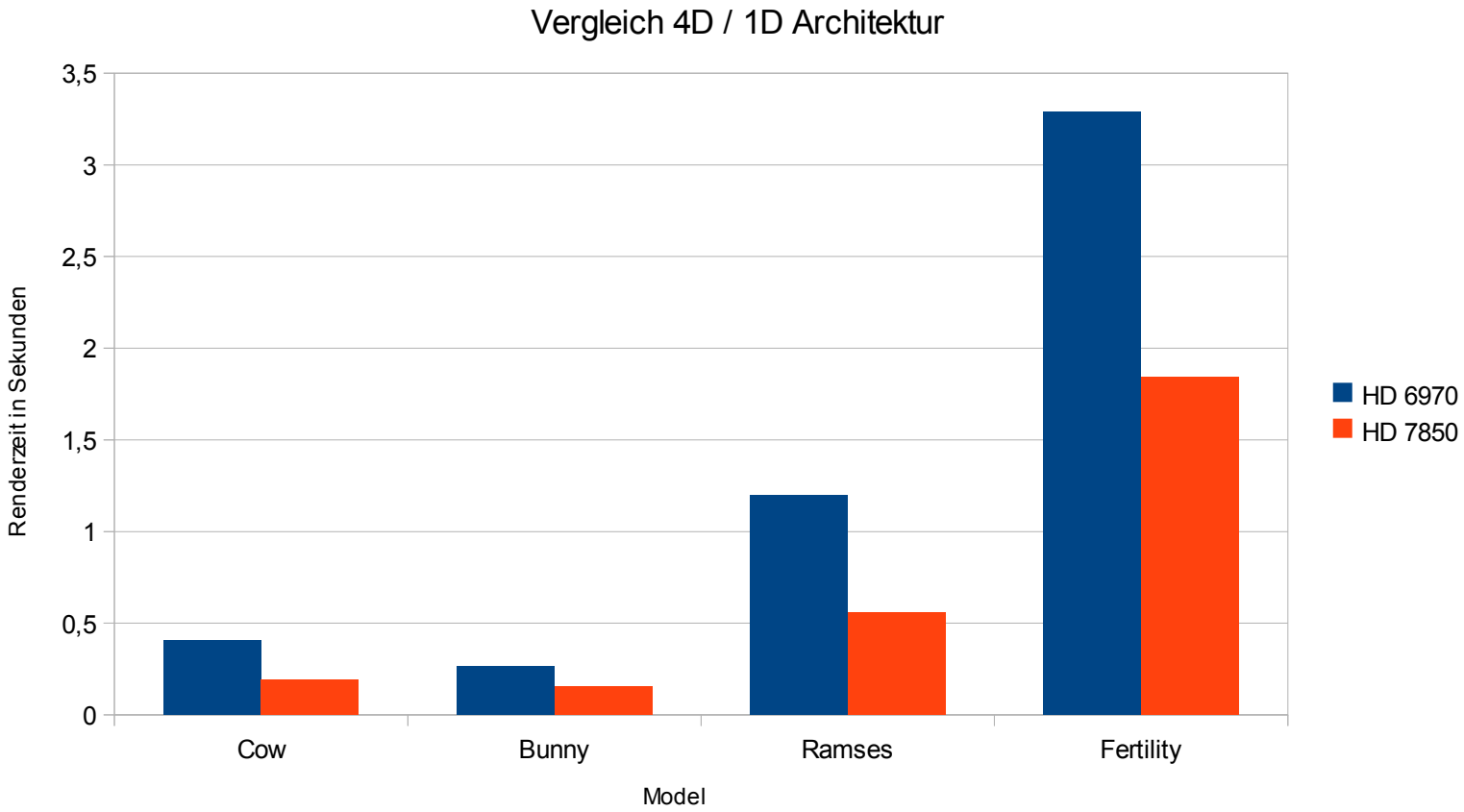
- Zur Veranschaulichung und zum Szenen-Erstellen
- Leistungsmessungen mittels Konsolen-Renderer
- Funktionalitäten für FPGA

Raytracing auf CPU / GPU



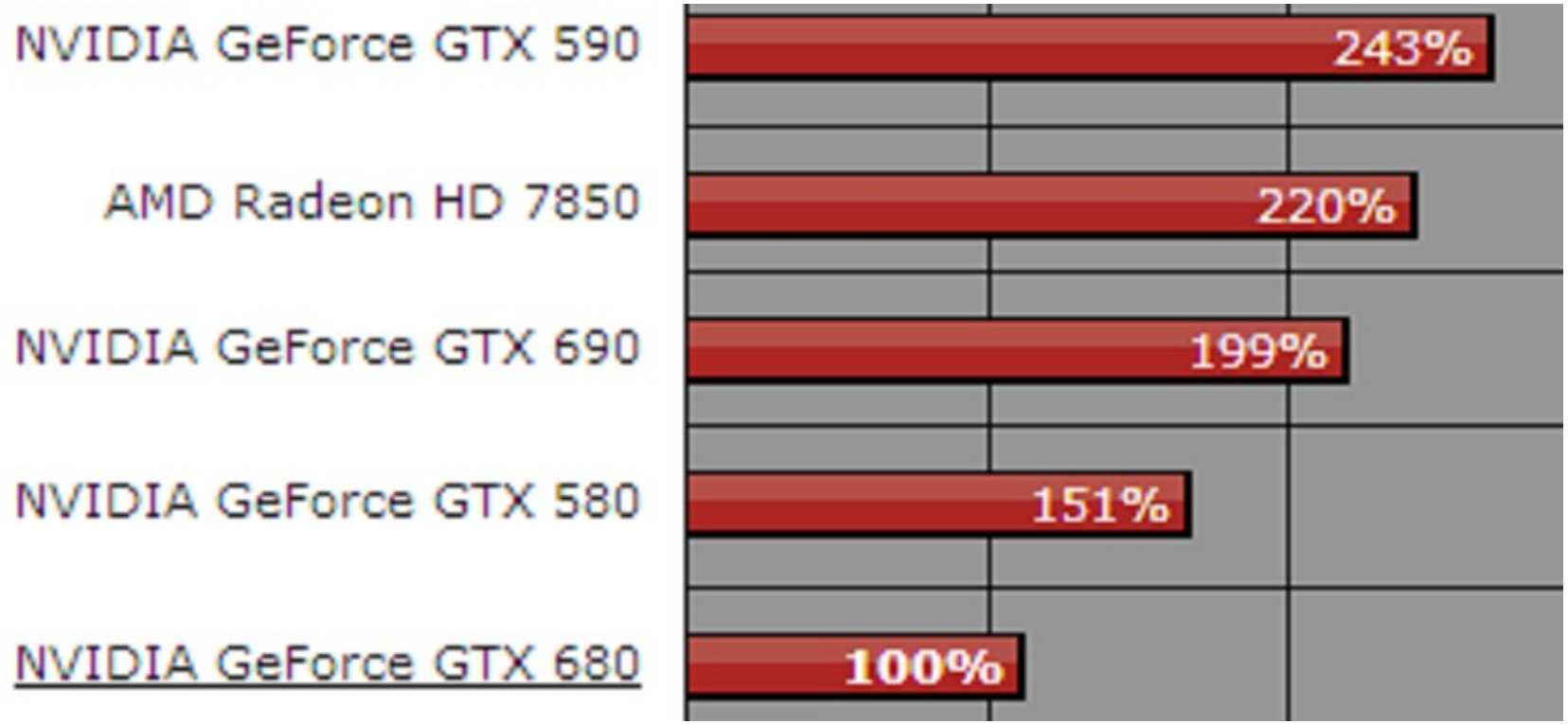
- Leistung auf CPU ~ Anzahl der Operationen
- GA nutzt Grafikhardware effektiver
- Differenz beim Raycasting etwas größer als beim Raytracing

Raytracing auf CPU / GPU



- Auf neuer Architektur deutlich effektiver berechenbar

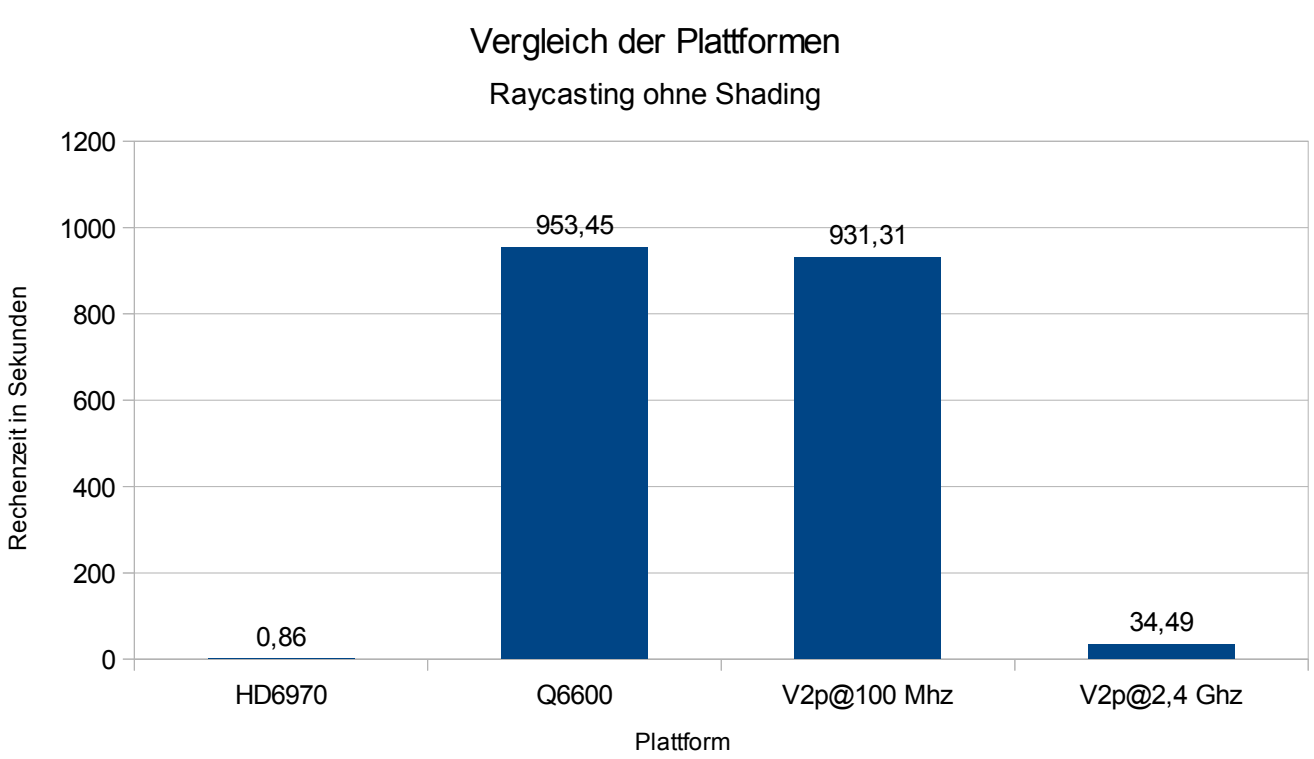
Raytracing auf CPU / GPU

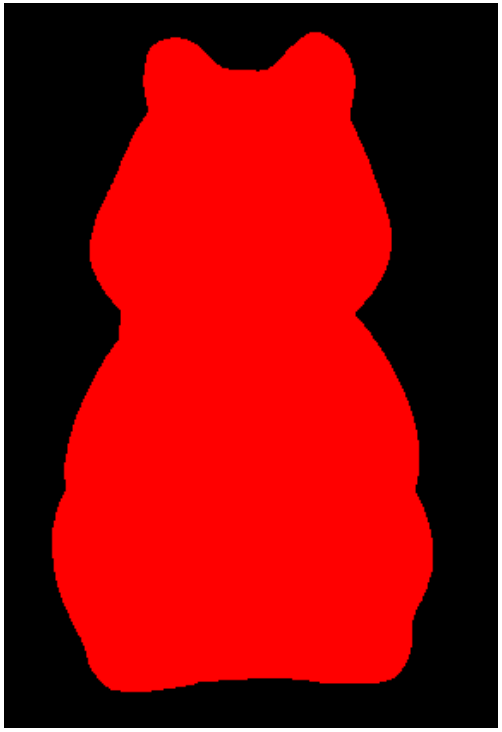


- AMDs HD7970 Ghz über viermal schneller als GTX680

Lux Renderer, LuxRay-Engine (OpenCL), Screenshot von www.ht4u.net

Raytracing auf CPU/GPU





Eine sehr kurze Einführung in OpenACC :)

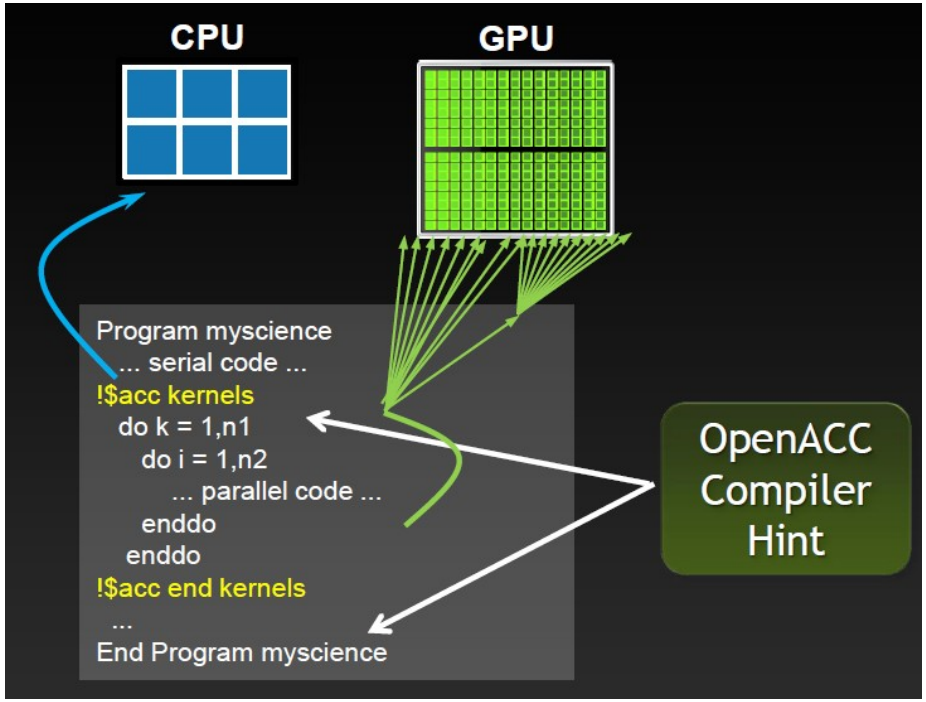
Was ist OpenACC?

TECHNISCHE UNIVERSITÄT DARMSTADT

OpenACC Standard

NVIDIA CRAY THE SUPERCOMPUTER COMPANY PGI CAPS

- Nutze Direktive im Quellcode
- Seriellen Code auf CPU
- Parallelen Code auf Accelerator



OpenACC Pragmas

TECHNISCHE UNIVERSITÄT DARMSTADT

 # pragma acc parallel [clause]

- Startet parallele Region
- Workers auf dem Accelerator werden initialisiert, Anzahl während der Region konstant

 # pragma acc kernels [clause]

- Compiler zerlegt den Code in Kernels, die auf Device ausgeführt werden
- Normalerweise erhält jede Schleife einen Kernel

#pragma acc loop [clause]

- Schleife wird parallelisiert

OpenACC Pragmas

TECHNISCHE UNIVERSITÄT DARMSTADT

if(*condition* **)**
async [**(** *scalar-integer-expression* **)**]
num_gangs(*scalar-integer-expression* **)**
num_workers(*scalar-integer-expression* **)**
vector_length(*scalar-integer-expression* **)**

OpenACC Data Pragmas

TECHNISCHE UNIVERSITÄT DARMSTADT

#pragma acc data [clause]

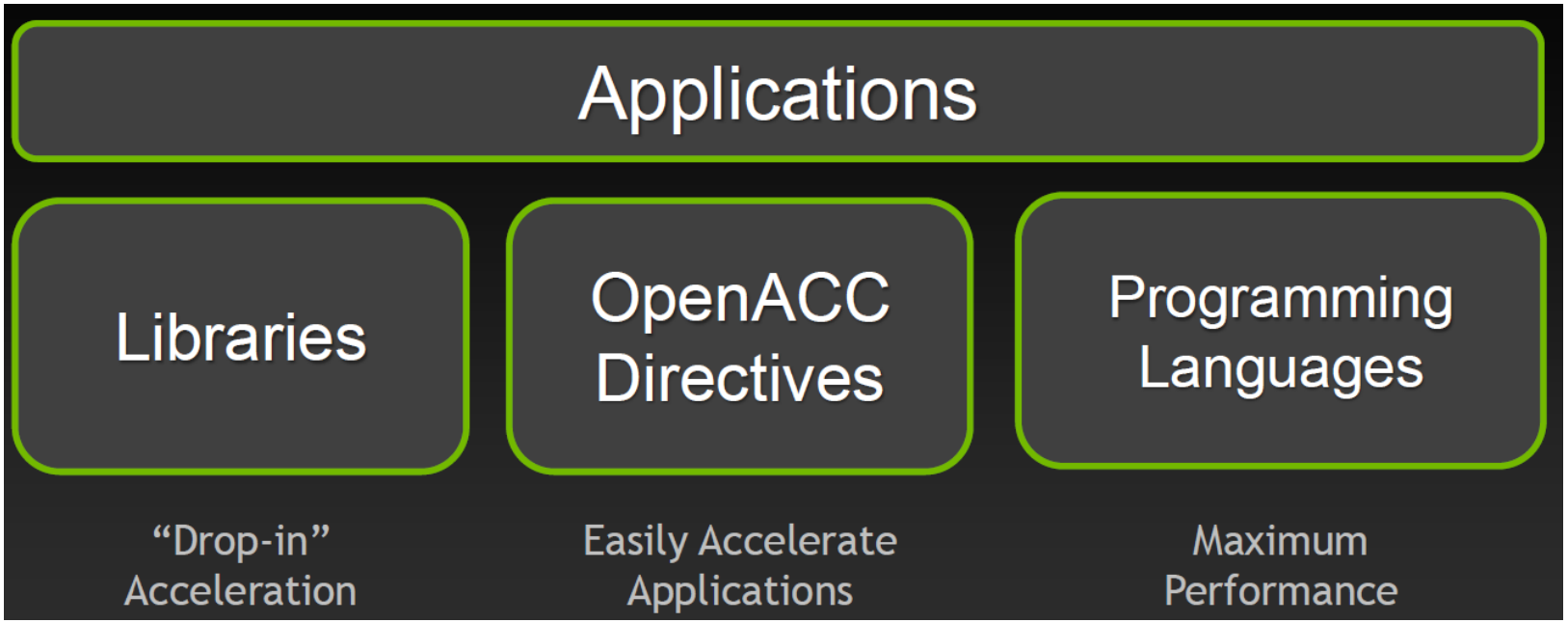
if(*condition* **)**
copyin(*list* **)**
create(*list* **)**
present_or_copy(*list* **)**
present_or_copyout(*list* **)**
deviceptr(*list* **)**

copy(*list* **)**
copyout(*list* **)**
present(*list* **)**
present_or_copyin(*list* **)**
present_or_create(*list* **)**

Position von OpenACC

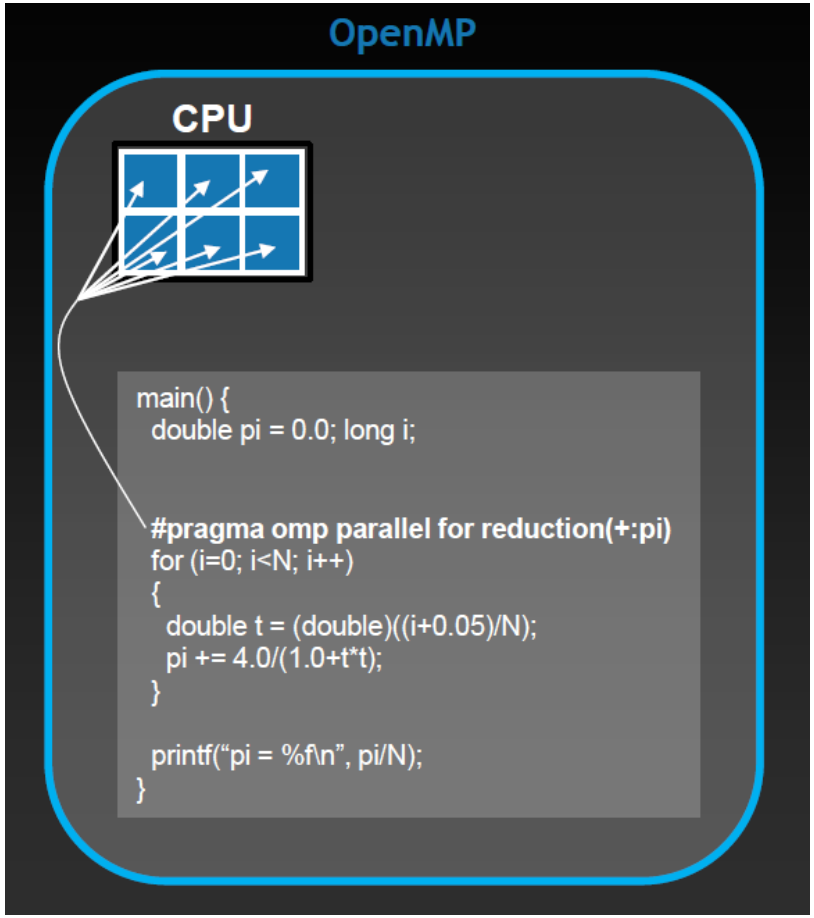
TECHNISCHE UNIVERSITÄT DARMSTADT

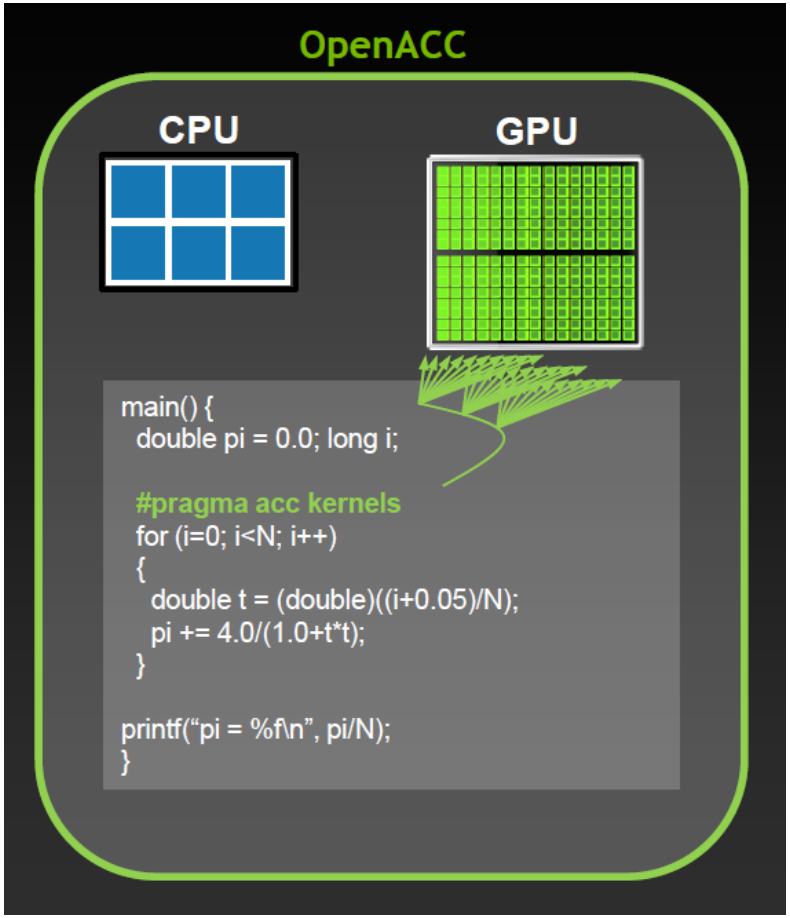
- Wie fügt sich OpenACC in bestehende Struktur?
- Flexibilität?
- Aufwand?



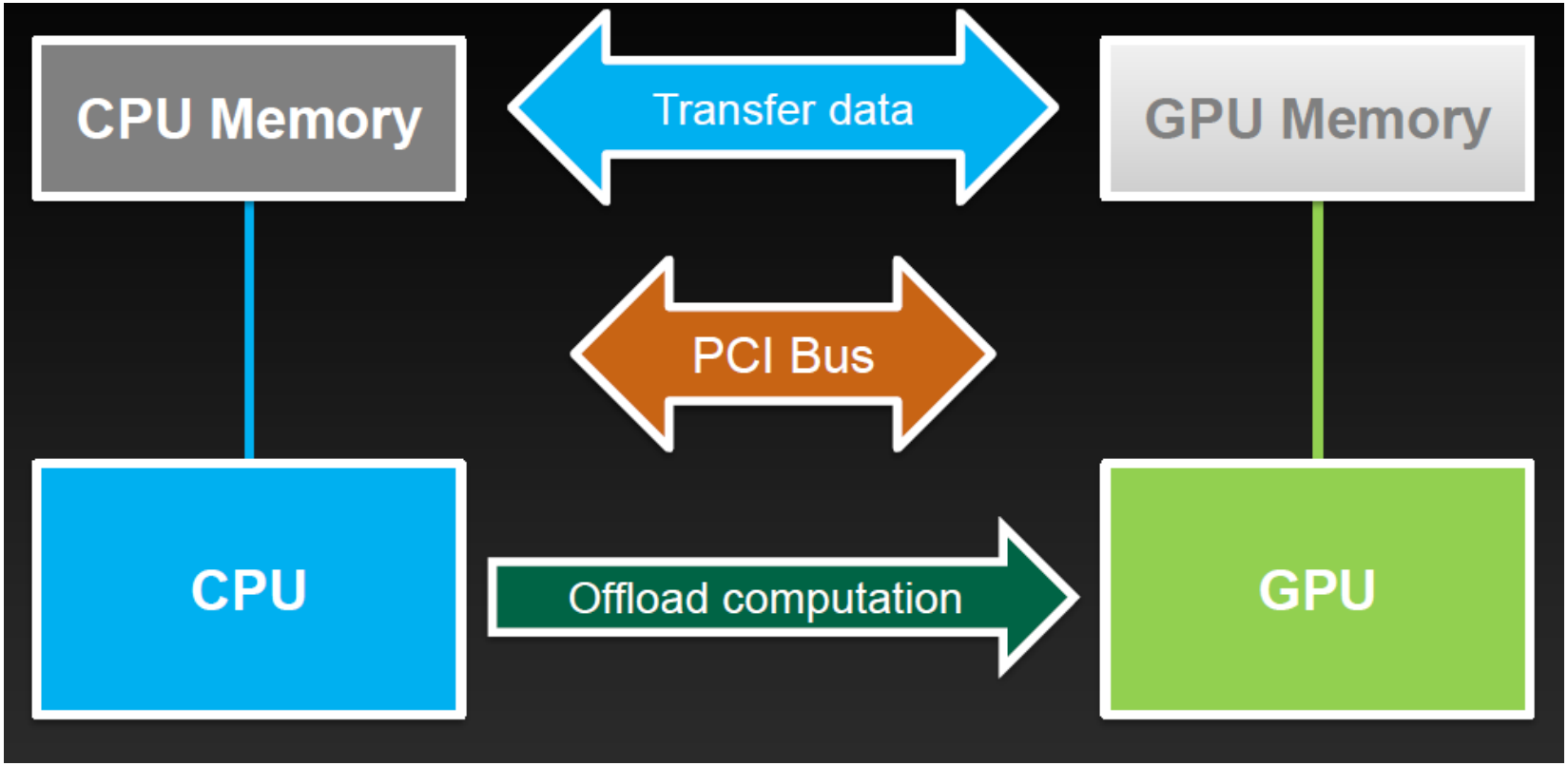
OpenACC und OpenMP

TECHNISCHE UNIVERSITÄT DARMSTADT





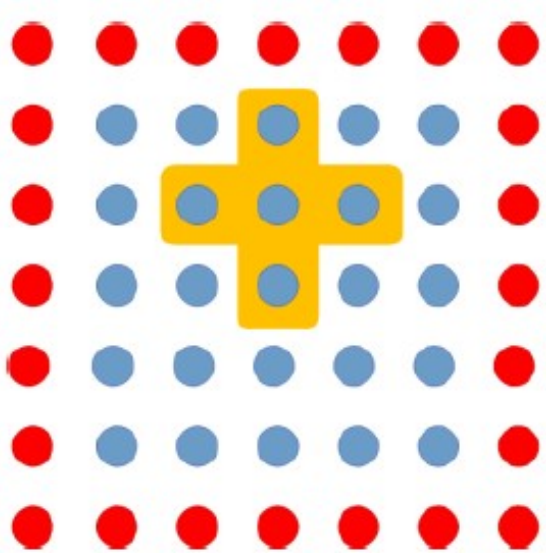
Schön und gut, aber...



- Datenmanagement sehr wichtig
- Datentransfer zeitintensiv

Beispiel: Jacobi Solver

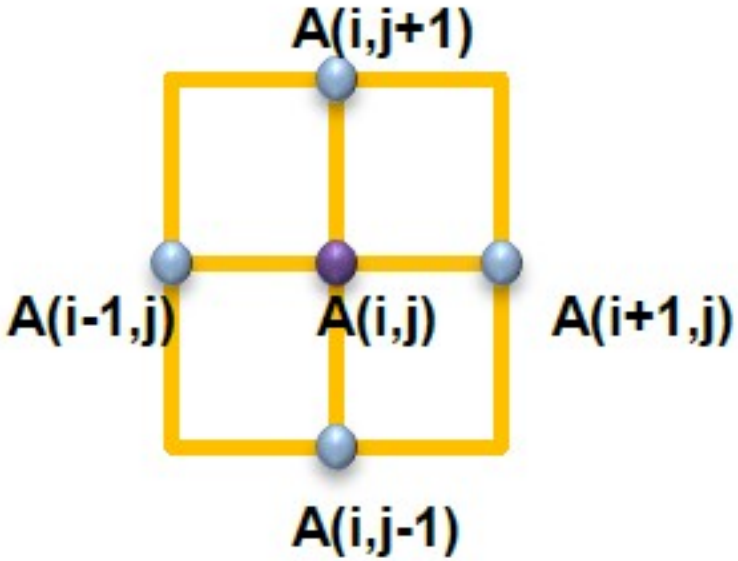
$$\nabla^2 f(x,y) = 0$$



Data Point

Boundary Point

Stencil



$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

Beispiel: Jacobi Solver

TECHNISCHE UNIVERSITÄT DARMSTADT

```
while ( error > tol && iter < iter_max )
{
    error = 0.0;

    for( int j = 1; j < n-1; j++)
    {
        for( int i = 1; i < m-1; i++ )
        {
            Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
        + A[j-1][i] + A[j+1][i]);
            error = fmax( error, fabs(Anew[j][i] - A[j][i]));
        }
    }

    for( int j = 1; j < n-1; j++)
    {
        for( int i = 1; i < m-1; i++ )
        {
            A[j][i] = Anew[j][i];
        }
    }
```

Beispiel: Jacobi Solver

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
            + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma acc kernels
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }
```


SCIENTIFIC COMPUTING

Beispiel: Jacobi Solver

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zeitverhalten der Version sehr schlecht
- Profiling ergibt viel Aufwand für Datentransfer
- Wenig Laufzeit für eigentliche Berechnungen

- **Ursache?**
- **Lösung?**

Beispiel: Jacobi Solver

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#pragma acc data copy(A), create(Anew)
    while ( error > tol && iter < iter_max )
    {
        error = 0.0;

#pragma acc kernels present(A,Anew)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
            + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }

#pragma acc kernels present(A,Anew)
        for( int j = 1; j < n-1; j++)
        {
            for( int i = 1; i < m-1; i++ )
            {
                A[j][i] = Anew[j][i];
            }
        }
```


SCIENTIFIC COMPUTING

Beispiel: Jacobi Solver

TECHNISCHE UNIVERSITÄT DARMSTADT

- Optimierung durch Kenntnis der Zielhardware
- Gang = #Cuda Blocks / Vector = #threads in Block
- Vector sollte Vielfaches von 32 sein
-

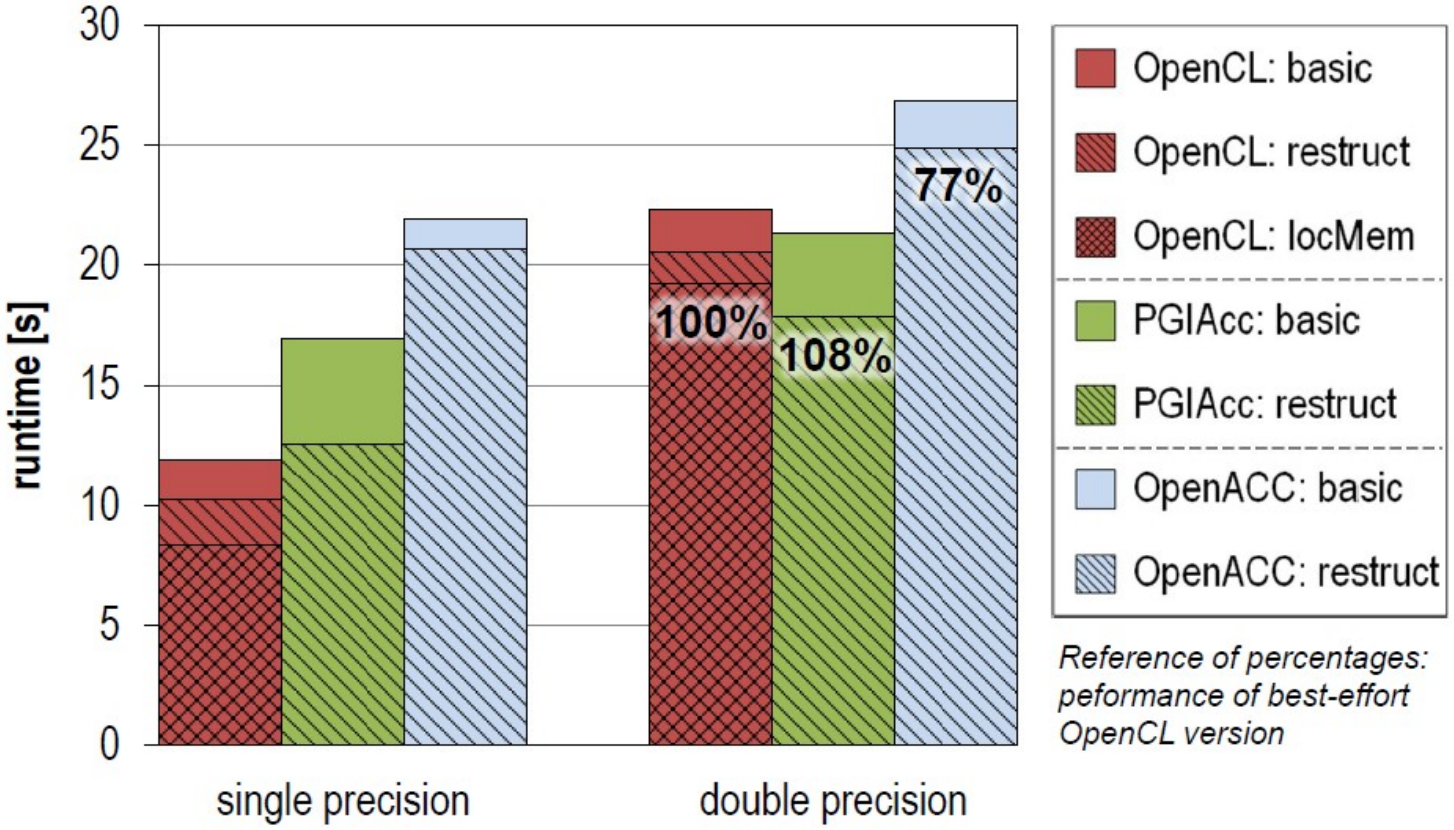
```
#pragma acc kernels present(A,Anew)
        for( int j = 1; j < n-1; j++)
        {
#pragma acc loop gang(16) vector(32)
            for( int i = 1; i < m-1; i++ )
            {
                Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1]
            + A[j-1][i] + A[j+1][i]);
                error = fmax( error, fabs(Anew[j][i] - A[j][i]));
            }
        }
```

Real-Wolrd Example OpenACC

TECHNISCHE UNIVERSITÄT DARMSTADT

RWTH AACHEN UNIVERSITY

Performance: *KegelSpan*



Real-World Example Raytracer?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Erzielbarer Performancegewinn im Vergleich zu OpenMP?
- Rückstand OpenCL-Variante auf gleicher Karte
- AMD OpenCL vs Nvidia OpenACC

Quellen

- Michael Burger, „Das effiziente Raytracen von Dreiecksnetzen auf Mehrkernprozessoren, GPUs und FPGAs mittels Geometrischer Algebra“, 2011
  http://www.gaalop.de/wp-content/uploads/Masterarbeit-Michael-Burger.pdf

- Timo Stich, „GPU Computing with OpenACC Directives“, 2012
  https://sharepoint.campus.rwth-aachen.de/units/rz/HPC/public/Shared%20Document

- Timo Stich & Sandra Wienke, „OpenACC Workshop – Programming Lab“, 2012
https://sharepoint.campus.rwth-aachen.de/units/rz/HPC/public/Shared%20Documents/

- Sandra Wienke et al, „OpenACC – First Experiences with Real-World Applications“, in Euro Par 2012

- OpenACC-Standard.Org, „The OpenACC Application Programming Interface“, 2011
- http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf