TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für technische Informatik

Evaluation der Tesla K20 (Kepler) Architektur und Erweiterung in CUDA 6.5

Kolloquium zur Projektarbeit ET-INF-D-900

Valentin Kandetzki
Valentin.Kandetzki@mailbox.tu-dresden.de

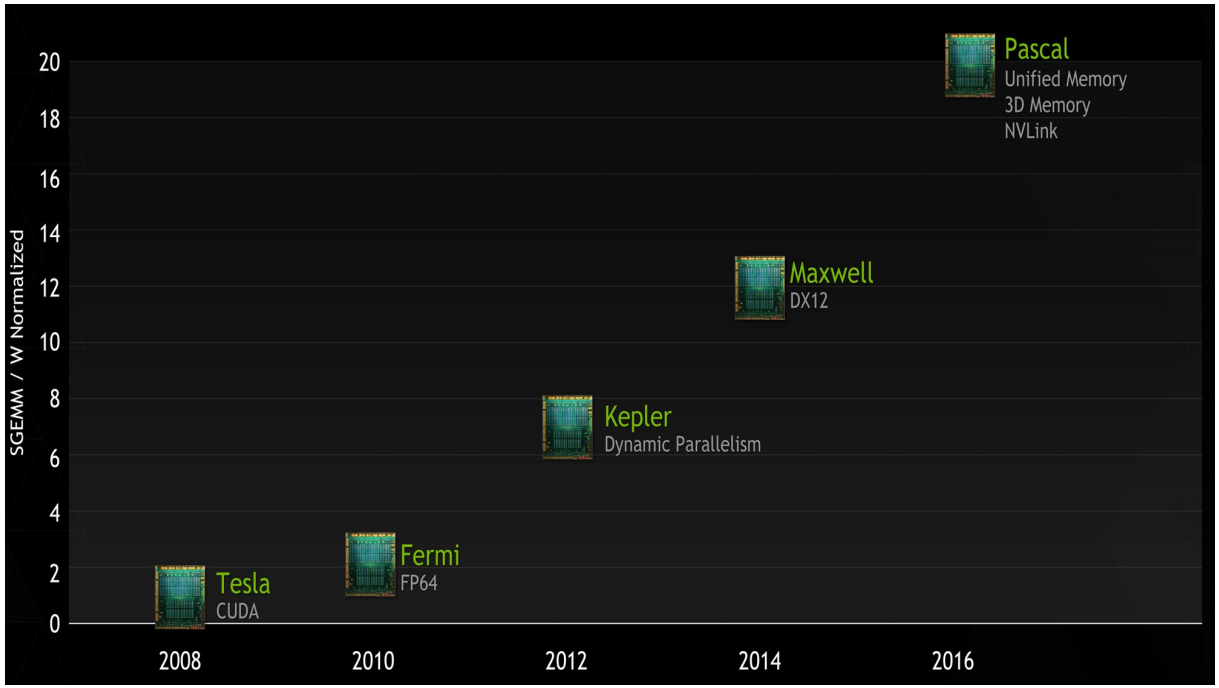
Dresden, 10.02.15

Gliederung

- 1. Kurze Geschichte
- 2. Compute Capability
- 3. Vorstellung der Tesla-Archtektur
- 4. Vorstellen der Neuerungen an CUDA 6.5
- 5. Testbeispiel
- 6. Zusammenfassung/ Ausblick
- 7. Quellen

1. Kurze Geschichte

- 2007: G80
- 2008: GT200
- 2009: Fermi
- 2012: Kepler
- 2014 Maxwell
- 2016 Pascal



[1]

Übersicht Chips

Chip	Stream Prozessoren	Fläche in mm^2	Transistoren in Mio.
G80	128	484	681
GT200	240	576	1400
Fermi	512	526	3000
Kepler GK104	1536	294	3540
Kepler GK110	2880	561	7100

Einordnung der K20

Modell	Prozessortyp	Rechen-leistung in GFlops	Speicher-größe in MB	Speichertyp	Speicher-durchsatz in GB/s
Tesla C870	G80	519	1536	GDDR3	77
Tesla C1060	GT200	SP: 936 DP: 78	4096	GDDR3	102
Tesla C2050	Fermi	SP:1030 DP: 515	3072	GDDR5	144
Tesla S2050	4x Fermi	SP: 4120 DP2060	12288	GDDR5	4x 144
Tesla K20	GK110	SP: 3520 DP: 1170	5120	GDDR5	208

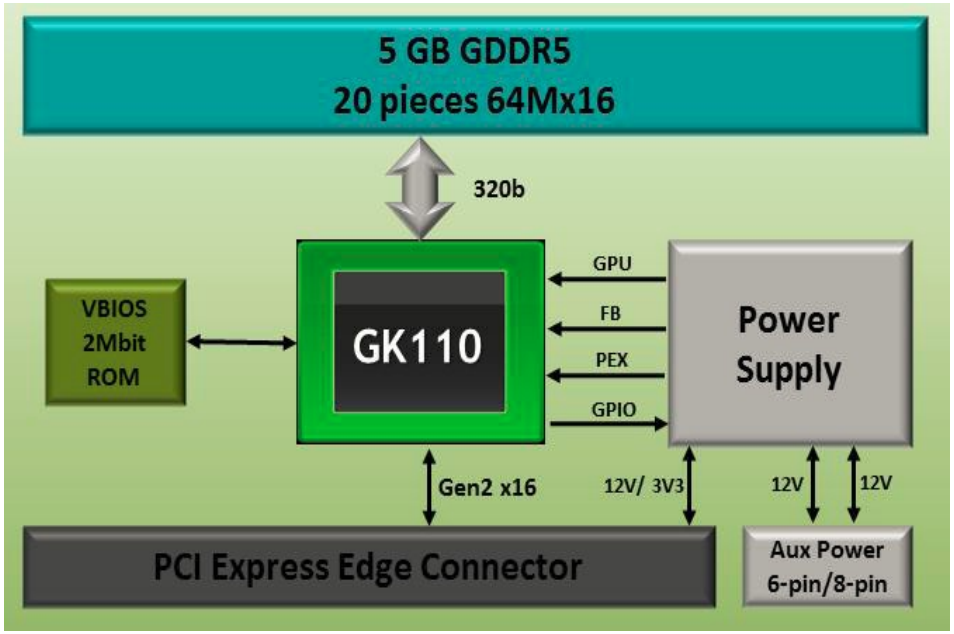
2. Compute Capability

- Gibt an, welche „Features“ auf einer speziellen Architektur verfügbar sind

- 1 – Tesla
- 2 – Fermi
- 3 – Kepler
- 5 – Maxwell

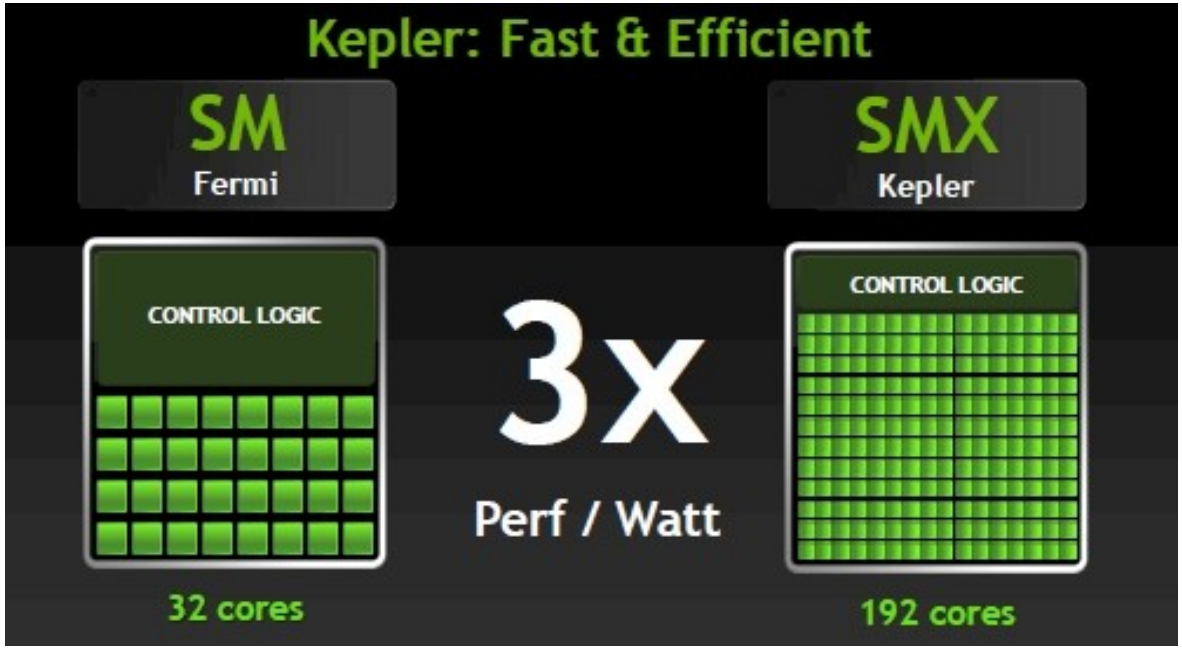
Feature support (unlisted features are supported for all compute capabilities)	Compute capability (version)								
	1.0	1.1	1.2	1.3	2.x	3.0	3.5	5.0	5.2
Integer atomic functions operating on 32-bit words in global memory	No	Yes							
atomicExch() operating on 32-bit floating point values in global memory									
Integer atomic functions operating on 32-bit words in shared memory	No	Yes							
atomicExch() operating on 32-bit floating point values in shared memory									
Integer atomic functions operating on 64-bit words in global memory									
Warp vote functions									
Double-precision floating-point operations	No				Yes				
Atomic functions operating on 64-bit integer values in shared memory	No				Yes				
Floating-point atomic addition operating on 32-bit words in global and shared memory									
_ballot()									
_threadfence_system()									
_syncthreads_count(), _syncthreads_and(), _syncthreads_or()									
Surface functions									
3D grid of thread block									
Warp shuffle functions	No					Yes			
Funnel shift	No						Yes		
Dynamic parallelism									

3. Die Tesla K20 Compute Capability 3.5



[3]

Größte Neuerungen - SMX



[4]

SM = SIMD Einheiten von nvidia

[5]

- 15 SMX-Multiprozessoren und 6 64-bit Memory Controller
- Multiprozessoren arbeiten unabhängig
- Die Grid Management Unit verteilt die Grids auf die SMX-Multiprozessoren

Fermi

[6]

Kepler

[7]

- 192 Single-Precision CUDA Cores
- 64 Double-Precision Einheiten
- 32 Special Function Einheiten (Inversion-, Inverse-Wurzel-, Logarithmus-, Exponential-, Sinus- und Cosinus-Funktionen)

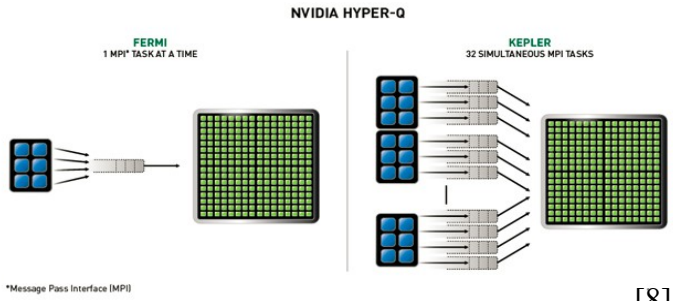
- 4 Warp Scheduler je SMX
  - Warp = Gruppe aus 32 parallelen Threads
  - Alle Warps teilen sich einen lokalen Speicher und einen Registersatz

- Maximal 64Wraps pro SMX

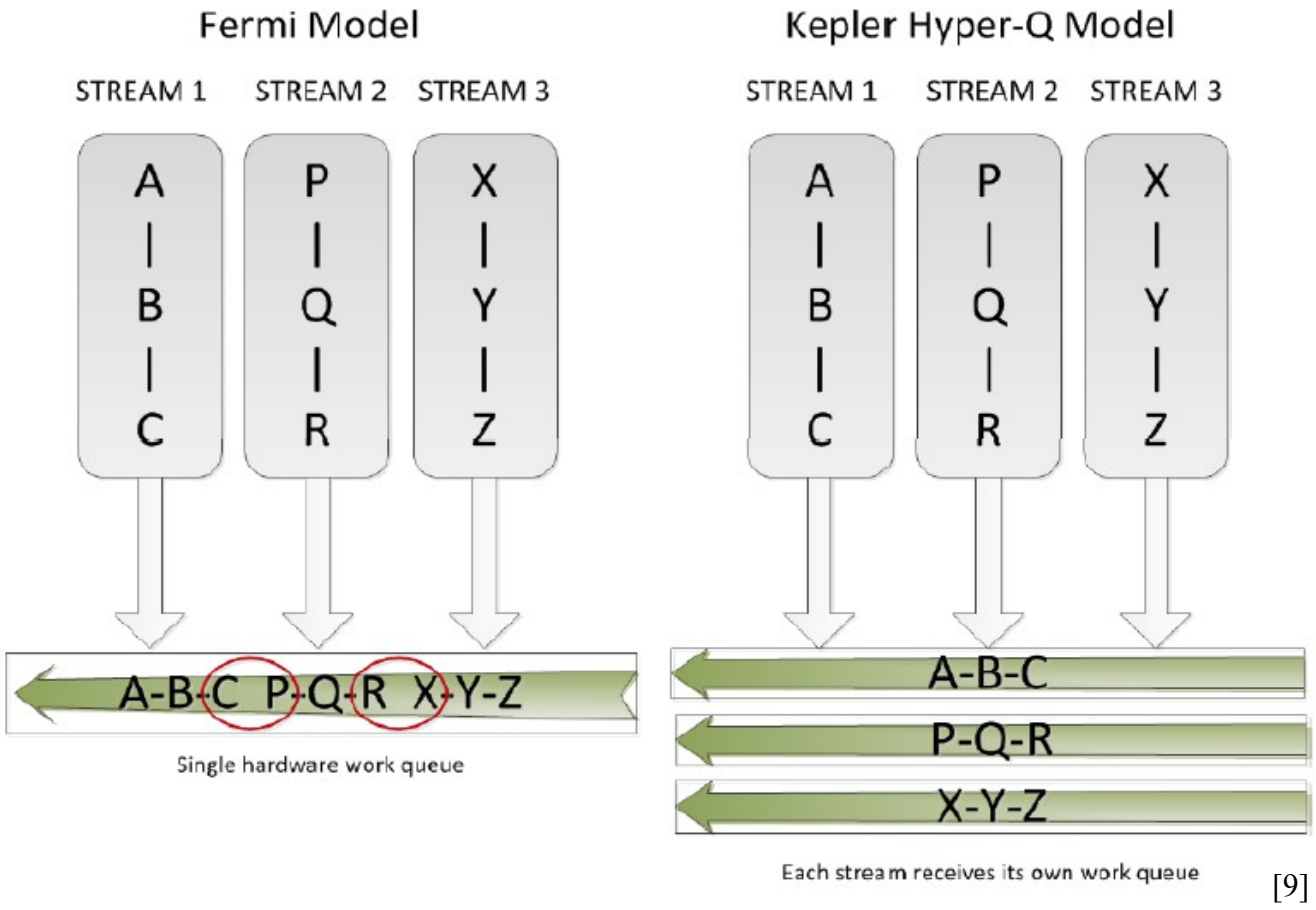
[7]

Größte Neuerungen - "Hyper-Q"

- GPU kann von 32 Threads einer CPU gleichzeitig versorgt werden
- 32 physikalische Kerne einer CPU können eine Kepler ansteuern
- Auch andersherum → CPU kann mit einem Thread mehrere GPUs ansteuern
- GPU Direct → GPUs können direkt auf Speicher anderer GPUs zugreifen

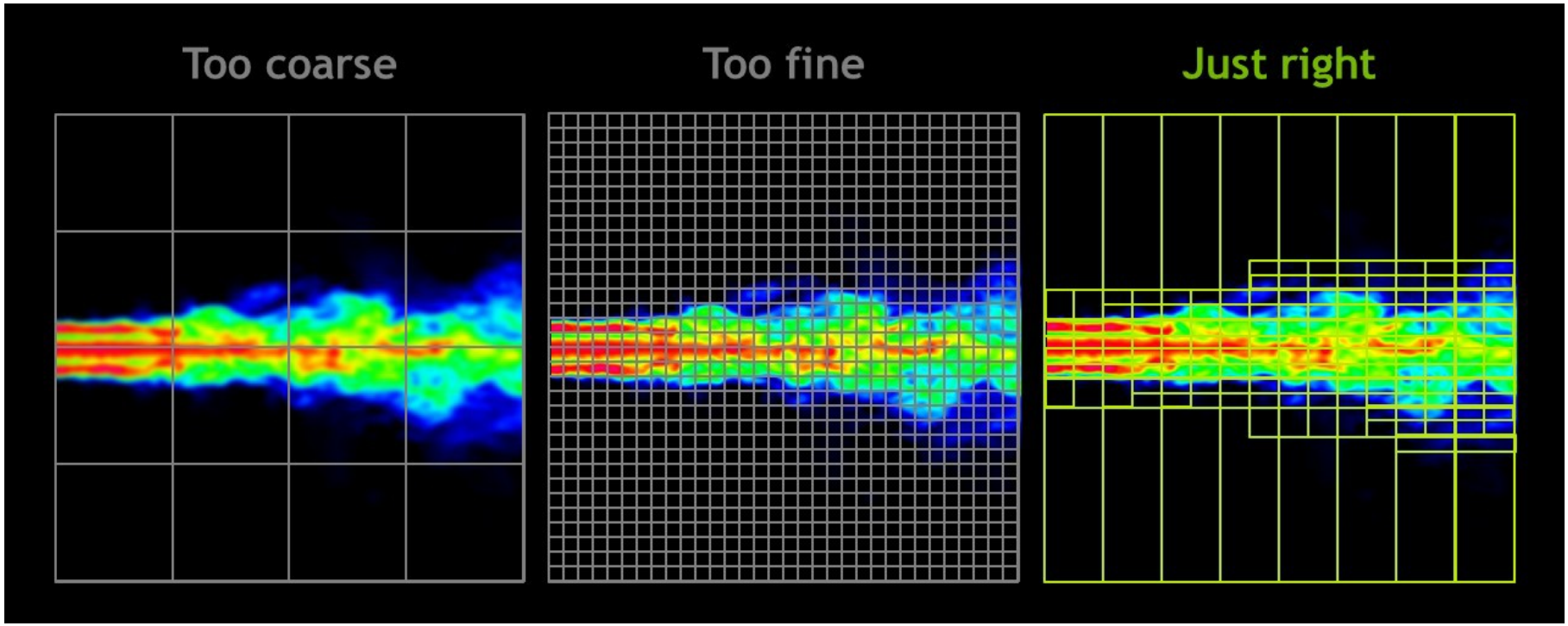


[8]


Fermi Model
STREAM 1
STREAM 2
STREAM 3
A
B
C
P
Q
R
X
Y
Z
A-B-C P-Q-R X-Y-Z
Single hardware work queue
Kepler Hyper-Q Model
STREAM 1
STREAM 2
STREAM 3
A
B
C
P
Q
R
X
Y
Z
A-B-C
P-Q-R
X-Y-Z
Each stream receives its own work queue

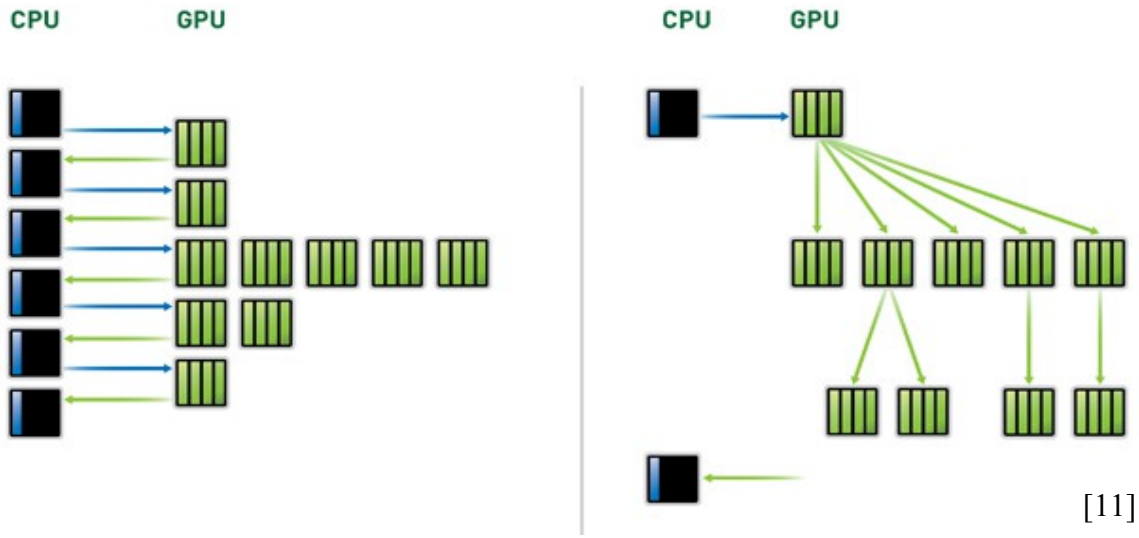
[9]

Größte Neuerungen – Dynamic Parallelism



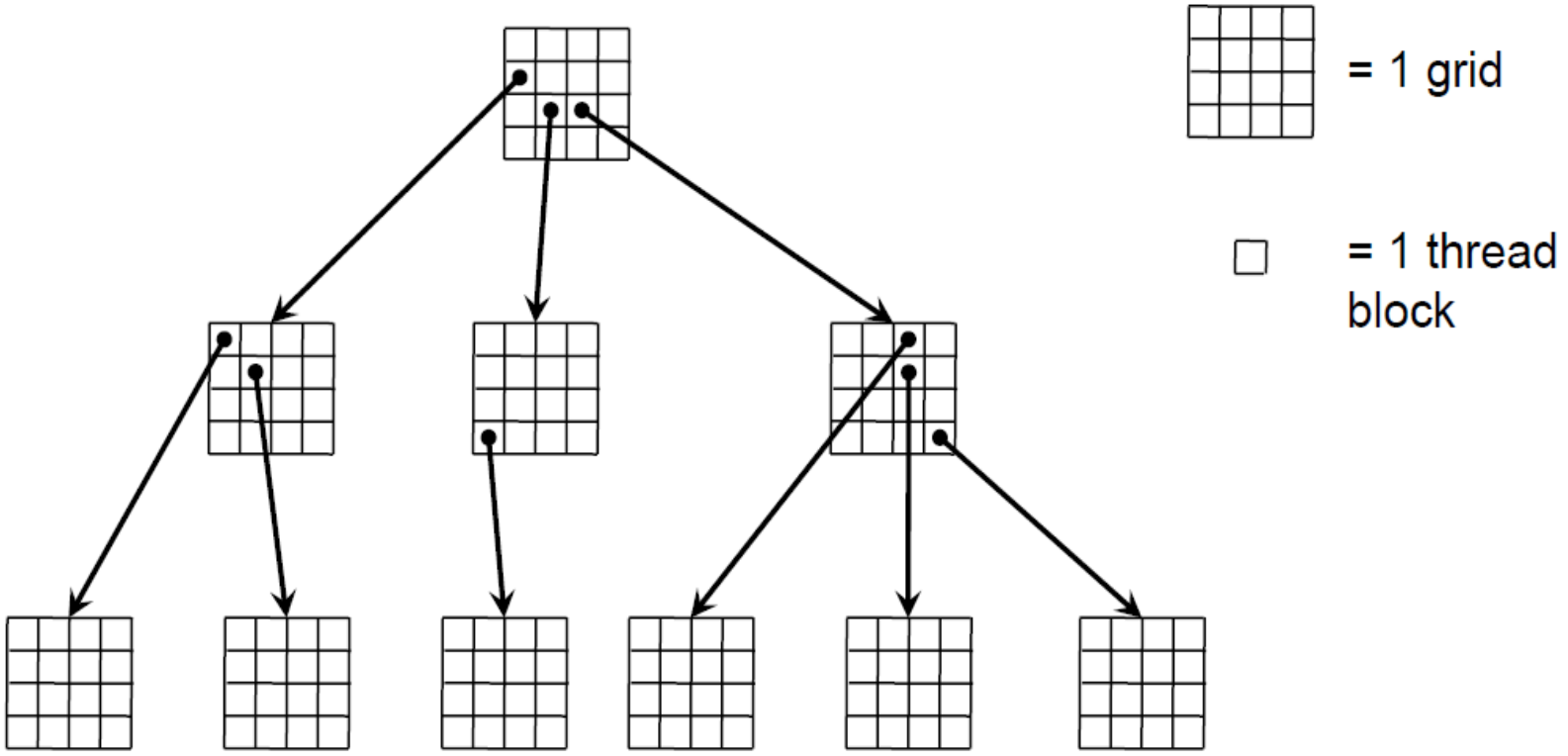
[10]

Größte Neuerungen – Dynamic Parallelism



[11]

- GPU kann mehrere Kernels und Threads von ihrem eigenem Speicher starten ohne CPU
- Um Speicherverwaltung muss sich selbst gekümmert werden



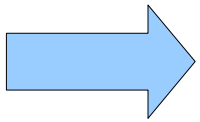
[12]

Codebeispiel

```
__global__ void childKernel(int parId){
    printf("Hello kid %d of parent %d\n", threadIdx.x, parId);
    }

__global__ void parentKernel(){
    childKernel<<<1,3>>>(threadIdx.x);
    cudaDeviceSynchronize();
    printf("Parent %d\n", threadIdx.x);
    }

int main(int argc, char* argv[]){
    parentKernel<<<1,5>>>();
    cudaDeviceSynchronize();
    return 0;
```

Codebeispiel

```
__global__ void childKernel(int parId){
    printf("Hello kid %d of parent %d\n", threadIdx.x, parId);
    }

__global__ void parentKernel(){
    childKernel<<<1,3>>>(threadIdx.x);
    cudaDeviceSynchronize();
    printf("Parent %d\n", threadIdx.x);
    }

int main(int argc, char* argv[]){
    parentKernel<<<1,5>>>();
    cudaDeviceSynchronize();
    return 0;
```



```
Hello kid 0 of parent 0
Hello kid 1 of parent 0
Hello kid 2 of parent 0
Hello kid 0 of parent 1
Hello kid 1 of parent 1
Hello kid 2 of parent 1
Hello kid 0 of parent 2
Hello kid 1 of parent 2
Hello kid 2 of parent 2
Hello kid 0 of parent 3
Hello kid 1 of parent 3
Hello kid 2 of parent 3
Hello kid 0 of parent 4
Hello kid 1 of parent 4
Hello kid 2 of parent 4
Parent 0
Parent 1
Parent 2
Parent 3
Parent 4
```

4. Vorstellen der Neuerungen an CUDA 6.5

- Unterstützung von 64-Bit-Systemen von ARM
  - ARM+K20 zeigt nahezu die gleiche Leistung wie x86+K20
- CUDA-Fortran-Werkzeuge (Debugger, Profiler, Memchecks)
- Cuda-Bibliothek für schnelle Fourier-Transformation
  - FFTs bis zu 10 mal schneller
- Neue Compiler-Zielplatform Visual Studio 2013

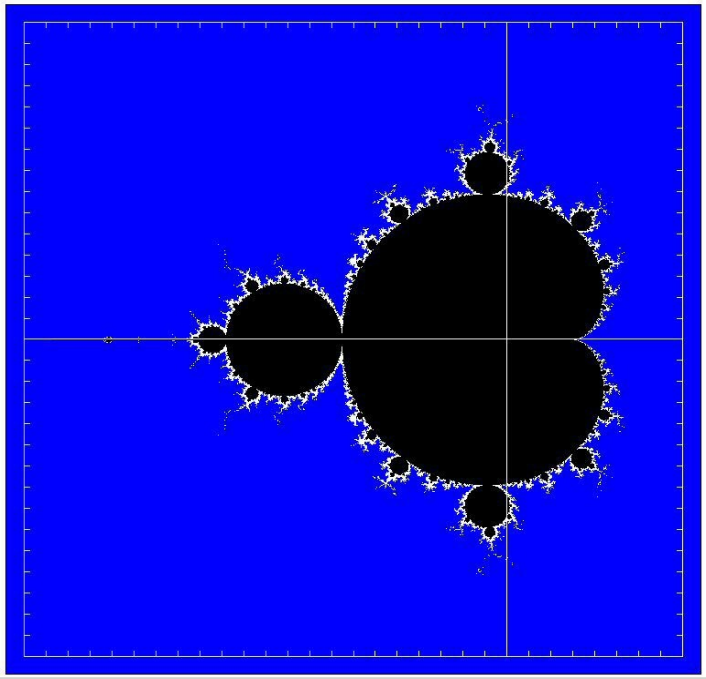
5. Testbeispiel – Mandelbrot Set

$$z_0 = c$$

$$z_{n+1} = z_n^2 + c$$

$$M = \{c \in \mathbf{C} : \exists R \; \forall n : z_n < R\}$$

Das beste bekannte Fraktal

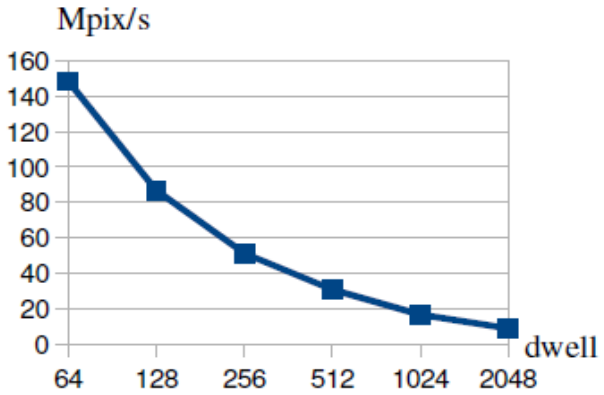


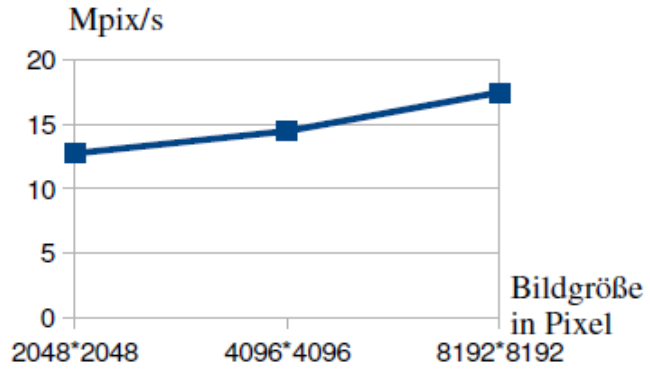
[13]

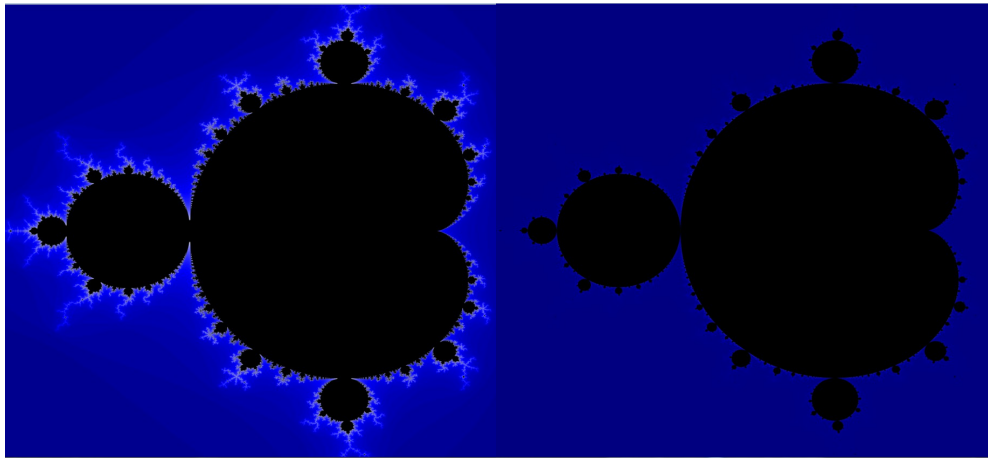
The Escape Time Algorithm – Pro-Pixel-Berechnung

```
#define MAX_DWELL 512
// w, h --- Breite und Höhe des Bildes in Pixel
// cmin, cmax --- Koordienaten der unteren linken und der oberen rechten Ecke
// x, y --- Koordinaten des Pixels
__host__ __device__ int pixel_dwell(int w, int h,
                                    complex cmin, complex cmax,
                                    int x, int y) {
  complex dc = cmax - cmin;
  float fx = (float)x / w, fy = (float)y / h;
  complex c = cmin + complex(fx * dc.re, fy * dc.im);
  complex z = c;
  int dwell = 0;

  while(dwell < MAX_DWELL && abs2(z) < 2 * 2) {
    z = z * z + c;
    dwell++;
  }

  return dwell;
}
```

Verhalten des Algorithmus bei unterschiedlicher Rekursionsanzahl und Bildgröße





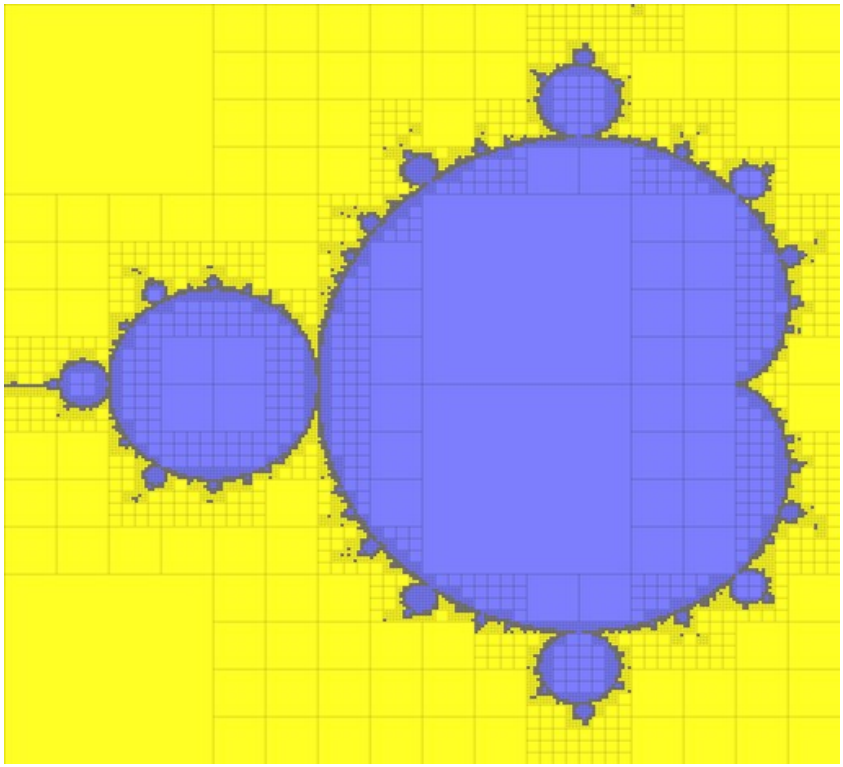


dwell = 64 dwell = 2048

Starker Geschwindigkeitsanstieg bei verringerter Rekursinszahl(dwell)

Leichter Anstieg bei erhöhung der Bildgröße

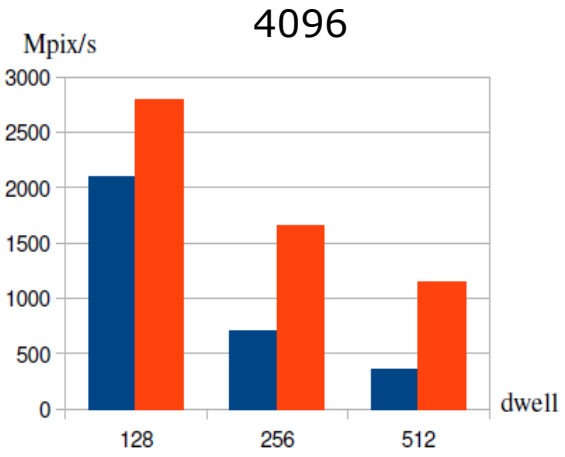
The Mariani-Silver Algorithm – Dynamic Parallelism

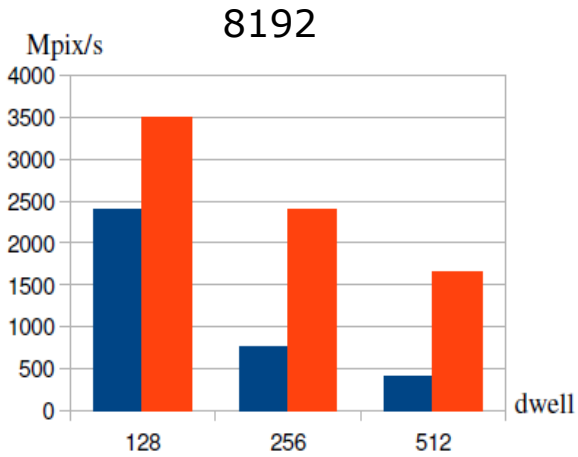


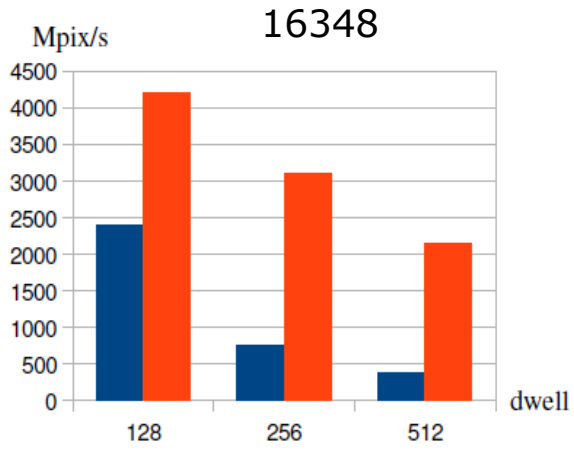
[14]

```
mariani_silver(Rechteck)
    if(grenze(Rechteck) hat gleichen dwell)
        fülle Rechteck mit gleichem dwell
    else if (Größe von Rechteck < treshold)
        pro Pixel Berechnung des Rechteckes
    else
        teile Rechteck in Teilrechtecke
        mariani_silver(Teilrechteck)
```

Vergleich Escape Time Algorithm und Mariani-Silver Algorithm







- Mpix/s in Abhängigkeit von dwell und Bildgröße
- Hoher dwell, großes Bild → größte Leistungssteigerung von Pro Pixel Berechnung zu Dynamic Parallelism
- Mit Erhöhung von dwell fällt Geschwindigkeit bei Dynamic Parallelism weniger ab als die bei Pro Pixel Perechnung

Zusammenfassung/ Ausblick

- Beginn der Tesla-Serie 2007 – starke Erhöhung an Cores
- Compute Capability um „Features" einer GPU zu bestimmen
- Die Kepler-Architektur → SMX, Hyper Q, Dynamic Parallelism
- Cuda 6.5 – neue Features
- Testbeispiel – starker Geschwindigkeitsgewinn mit Dynamic Parallelism

- Maxwell Architektur
  - Noch nicht im HPC Bereich – weniger DB Units

- Pascal Architektur
  - 3D Speicher → starker Anstieg der Speicherbandbreite auf ca. 1000GB/s
  - NVLink → Steigerung der Bandbreite zur CPU um ca. das 5- bis 12-Fache
  - Anwendung im HPC Bereich

7. Quellen

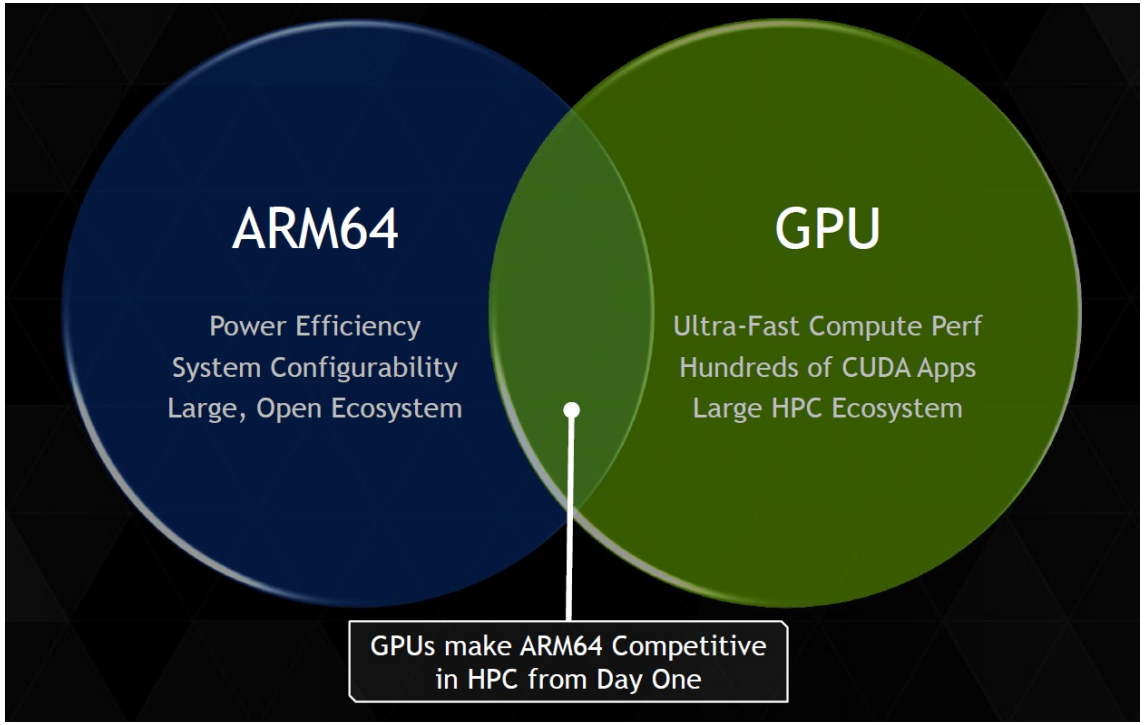
- http://www.extremetech.com/wp-content/uploads/2013/02/GK110_SMX_Diagram_FIN
- http://www.nvidia.com/content/kepler-compute-architecture/images/emeai/large-vide
- http://www.nvidia.com/content/kepler-compute-architecture/images/emeai/large-vide
- http://horstth.de/wp-content/uploads/2013/09/Apfelmann05.jpg
- http://devblogs.nvidia.com/parallelforall/introduction-cuda-dynamic-parallelism/
- https://developer.nvidia.com/cuFFT
- http://de.wikipedia.org/wiki/Nvidia_Tesla
- http://www.nvidia.de/object/tesla-server-gpus-de.html
- http://www.golem.de/news/kepler-gk110-groesster-chip-der-welt-mit-7-mia-transisto
- http://www.golem.de/news/programmierschnittstelle-nvidias-cuda-6-5-fuer-mehr-tem
- http://en.wikipedia.org/wiki/GeForce#PASCAL
- http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabili

7. Quellen

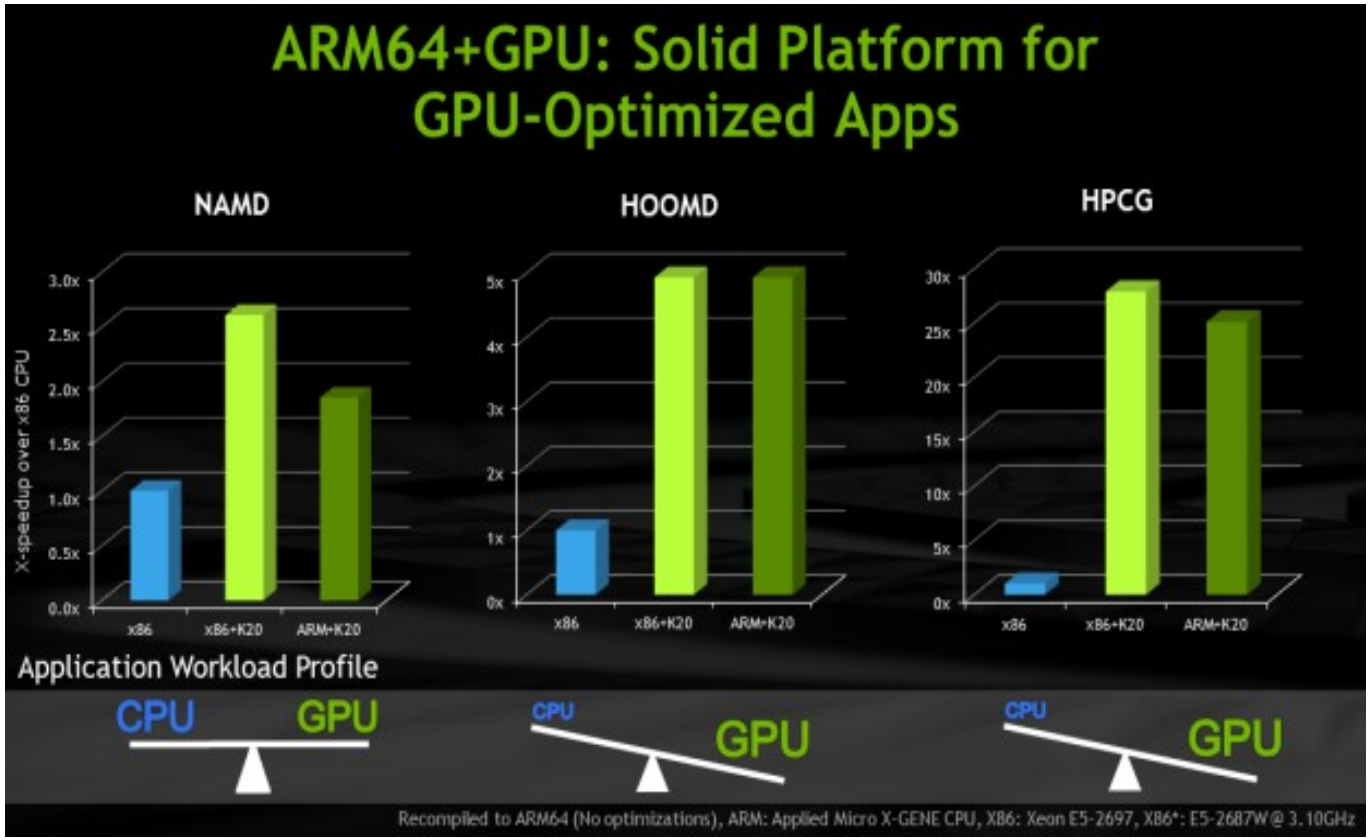
- [1] http://images.anandtech.com/doci/7900/PascalRoadmap.jpg
- [2] http://en.wikipedia.org/wiki/CUDA
- [3] http://www.nvidia.de/content/PDF/kepler/Tesla-K20-Passive-BD-06455-001-v07.pdf
- [4] http://assets.hardwarezone.com/img/2012/08/SMX_M.jpg
- [5] http://cdn.wccftech.com/wp-content/uploads/2012/10/GK110-Tesla-K20-Block-Diagram.jpg
- [6] http://www.geeks3d.com/public/jegx/201001/fermi_gt100_sm.jpg
- [7] http://img.hexus.net/v2/lowe/News/NVIDIA/k20.jpg
- [8] http://www.nvidia.com/content/kepler-compute-architecture/images/emeai/large-video-hyper-q-1-en.jpg
- [9] http://images.anandtech.com/doci/6446/HyperQ.png
- [10] http://3dgep.com/wp-content/uploads/2012/10/Dynamic-Parallelism-2.png
- [11] http://www.nvidia.com/content/kepler-compute-architecture/images/emeai/large-video-dynamic-parallelism-1-en.jpg
- [12] http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/advanced-gpu/adv-gpu-dynpar.html
- [13] http://horstth.de/wp-content/uploads/2013/09/Apfelmann05.jpg
- [14] http://devblogs.nvidia.com/parallelforall/wp-content/uploads/sites/3/2014/04/mariani-silver-subdivision.png

»Wissen schafft Brücken.«

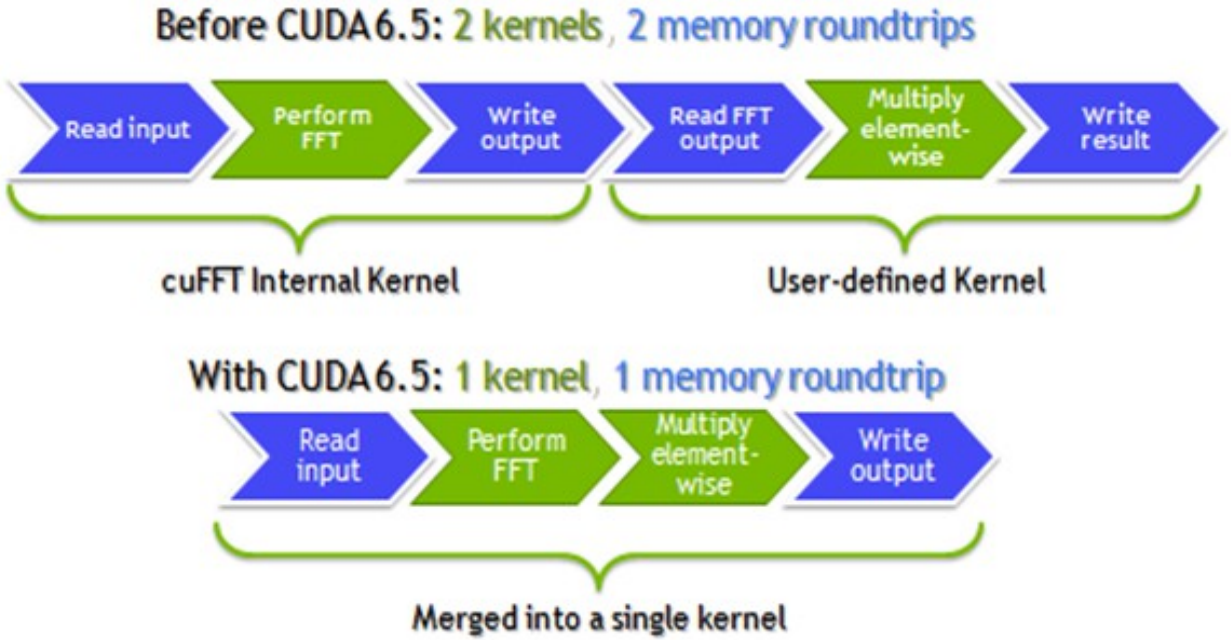
4. Vorstellen der Neuerungen an CUDA 6.5



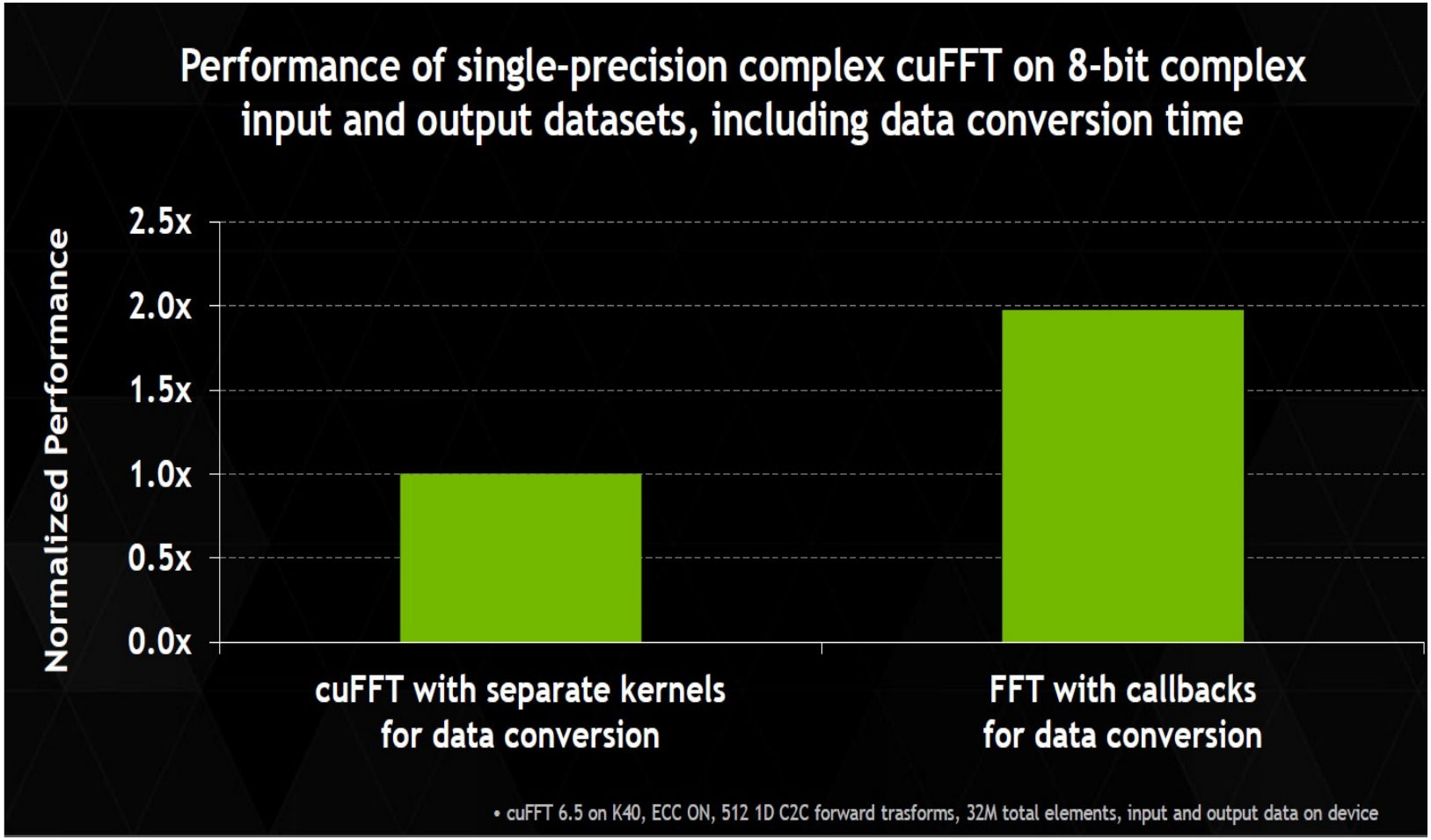
- Unterstützung für 64-Bit-Systeme von ARM


ARM64+GPU: Solid Platform for GPU-Optimized Apps
NAMD
HOOMD
HPCG
X-speedup over x86 CPU
3.0x
2.5x
2.0x
1.5x
1.0x
0.5x
0.0x
5x
4x
3x
2x
1x
0x
30x
25x
20x
15x
10x
5x
0x
x86
x86+K20
ARM+K20
Application Workload Profile
CPU
GPU
Recompiled to ARM64 (No optimizations), ARM: Applied Micro X-GENE CPU, X86: Xeon E5-2697, X86*: E5-2687W @ 3.10GHz

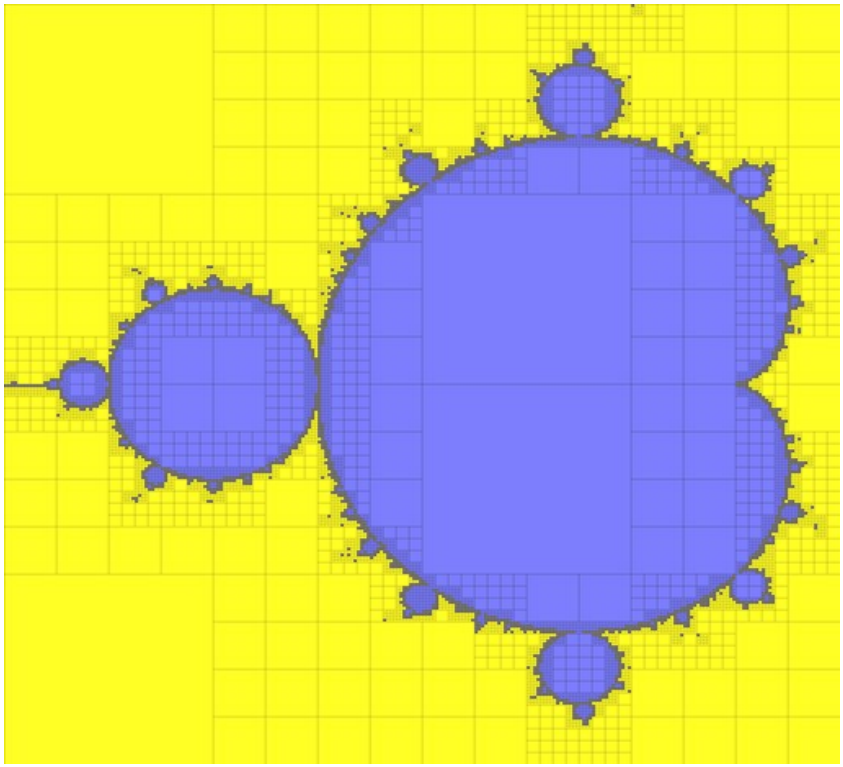
4. Vorstellen der Neuerungen an CUDA 6.5



- CUDA-Bibliothek für schnelle Fourier-Transformation


Performance of single-precision complex cuFFT on 8-bit complex input and output datasets, including data conversion time
Normalized Performance
2.5x
2.0x
1.5x
1.0x
0.5x
0.0x
cuFFT with separate kernels for data conversion
FFT with callbacks for data conversion
• cuFFT 6.5 on K40, ECC ON, 512 1D C2C forward trasforms, 32M total elements, input and output data on device

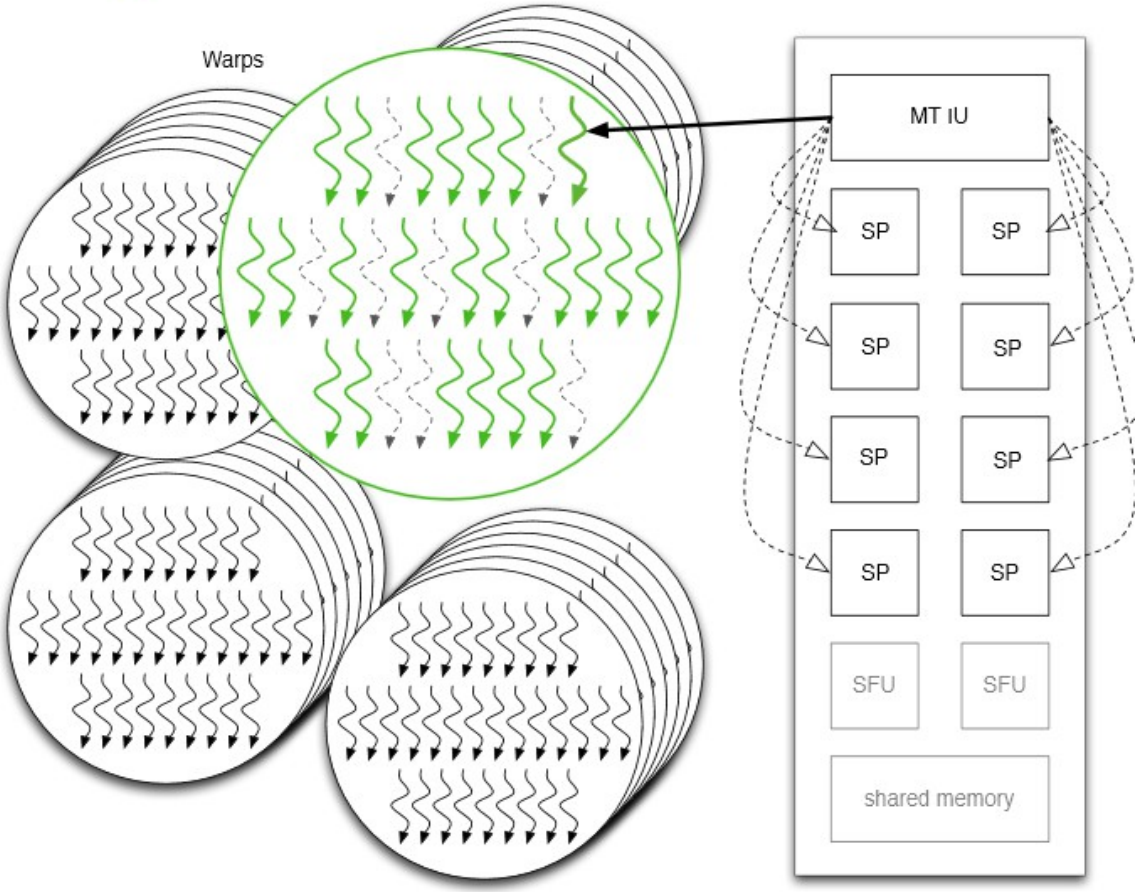
Gleiche Farbe?

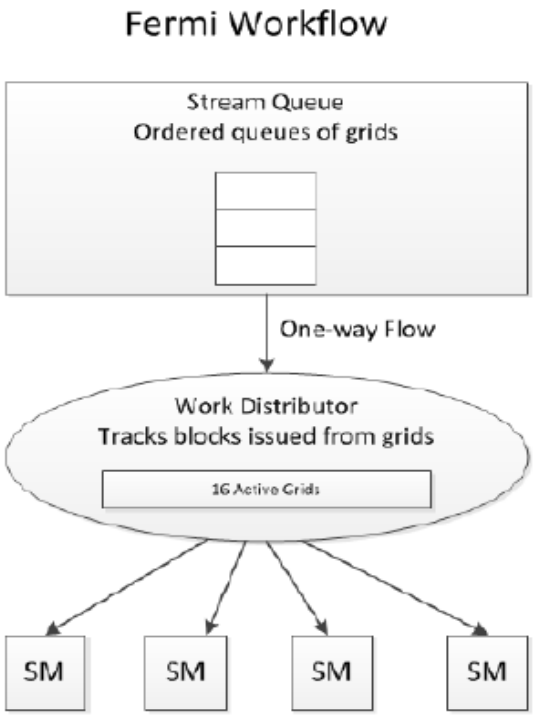


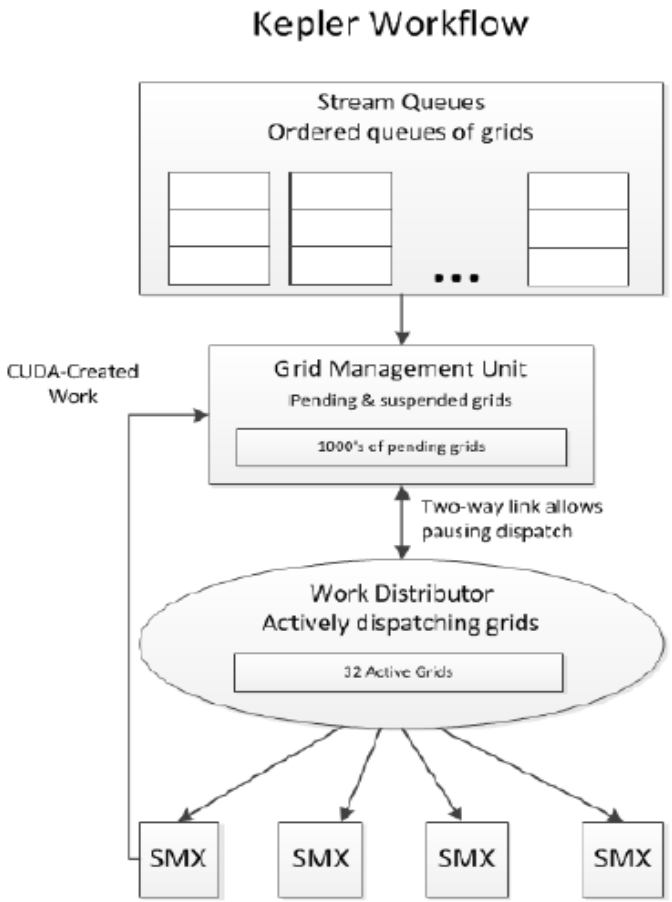
```
#define NEUT_DWELL (MAX_DWELL + 1)
#define DIFF_DWELL (-1)

__device__ int same_dwell(int d1, int d2) {
  if (d1 == d2)
    return d1;
  else if (d1 == NEUT_DWELL || d2 == NEUT_DWELL)
    return min(d1, d2);
  else
    return DIFF_DWELL;
}
```

Hardware Multithreading

3. Instruktion dieses Threads wird für den gesamten Warp ausgeführt
   - 3.1. Threads, welche die Instruktion ausführen müssen, beteiligen sich
   - 3.2. Threads, welche die Instruktion nicht ausführen müssen, warten




Fermi Workflow
Stream Queue
Ordered queues of grids
One-way Flow
Work Distributor
Tracks blocks issued from grids
16 Active Grids
SM
SM
SM
SM


Kepler Workflow
Stream Queues
Ordered queues of grids
CUDA-Created Work
Grid Management Unit
Pending & suspended grids
1000's of pending grids
Two-way link allows pausing dispatch
Work Distributor
Actively dispatching grids
32 Active Grids
SMX
SMX
SMX
SMX