TECHNISCHE UNIVERSITÄT DRESDEN

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

8. HPC-Status-Konferenz der Gauß-Allianz in Erlangen

MEPHISTO:

Metaprogrammierung für Heterogene Verteilte Systeme

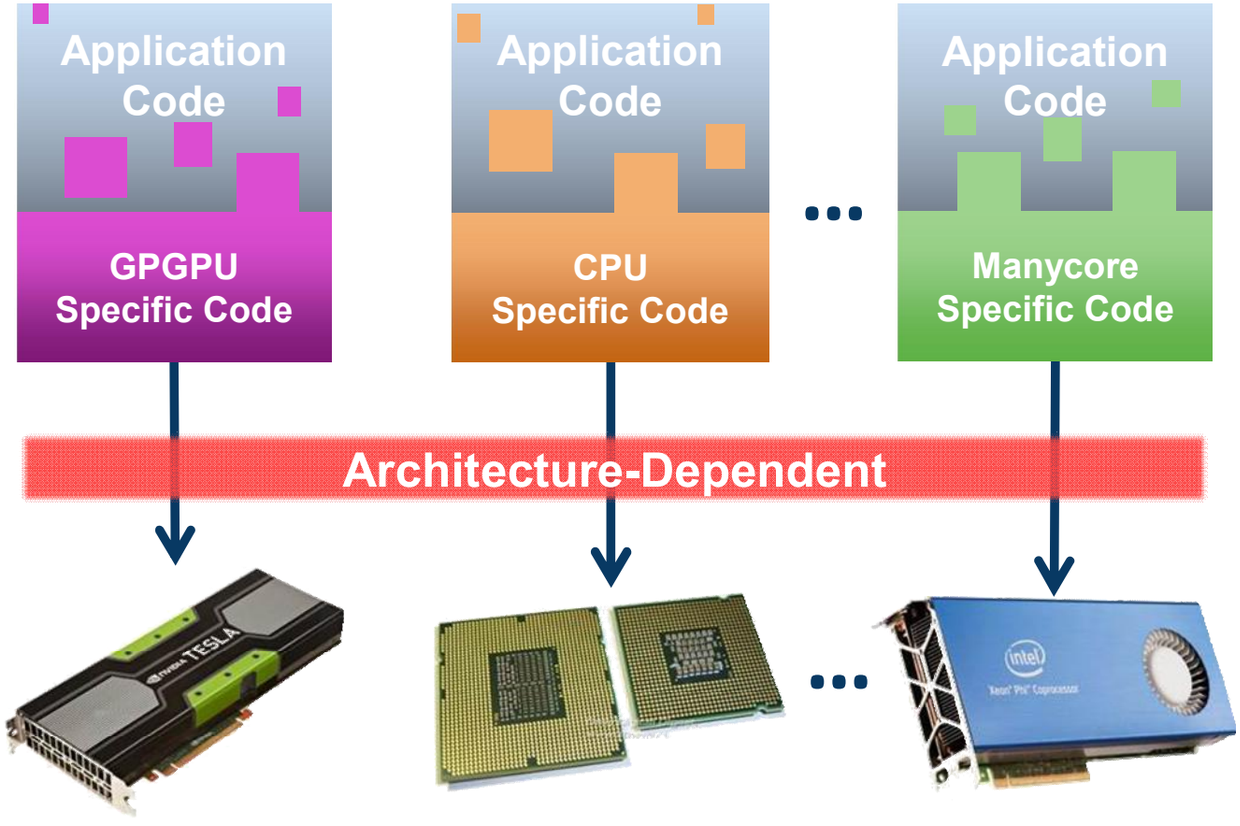
Dr. Karl Fürlinger, LMU München
Dr. Andreas Knüpfer, Bert Wesarg, TU Dresden
With material from: A. Matthes, R. Widera & M. Bussmann, HZDR, P. Jungblut, LMU

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)	
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,282,544	122,300.0	187,659.3	8,806	GPU
2	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371	
3	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/NNSA/LLNL United States	1,572,480	71,610.0	119,193.6		GPU
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482	
5	**AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2550 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	391,680	19,880.0	32,576.6	1,649	GPU
6	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272	GPU
7	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209	GPU
8	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890	
9	**Trinity** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,968	14,137.3	43,902.6	3,844	Xeon Phi
10	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939	Xeon Phi

Motivation: Top 500 Liste (06/2018)

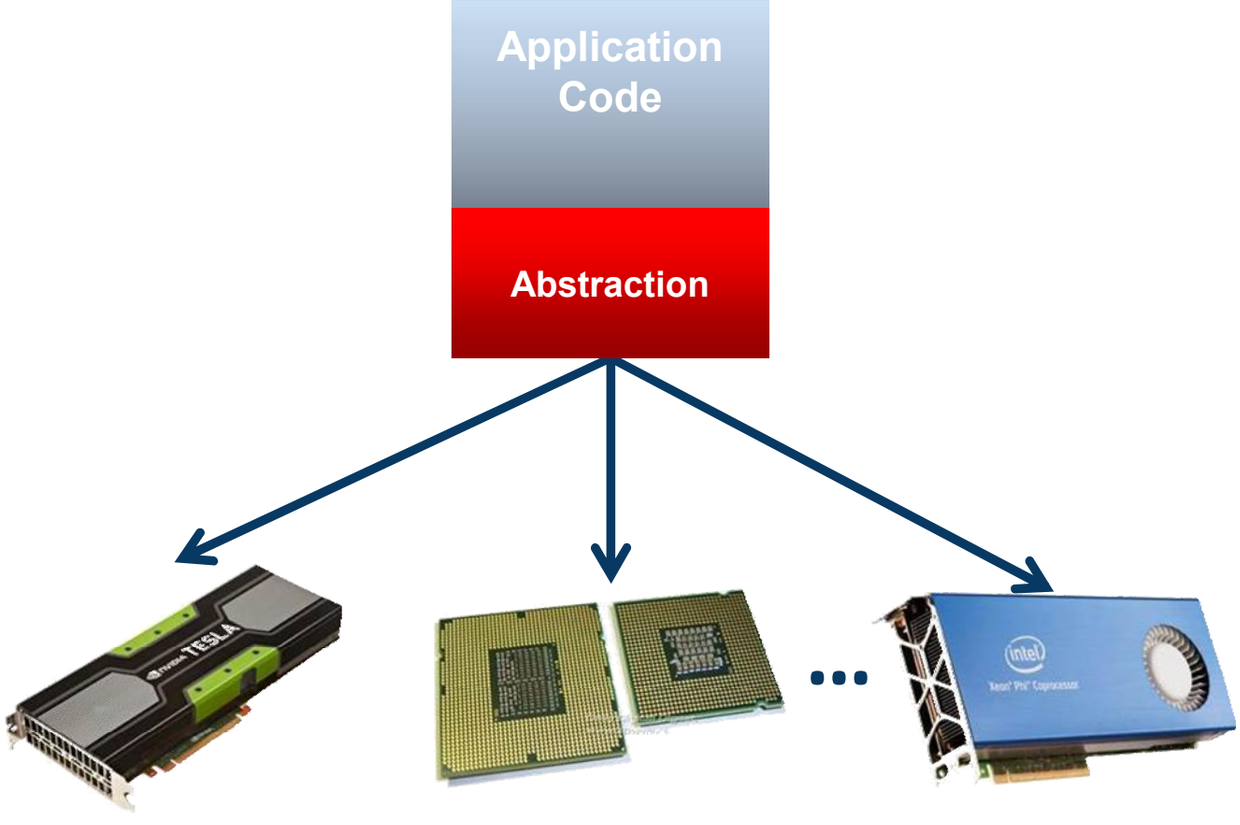
- Große Diversität im Design der Systeme
- Überwiegend GPU- und Manycore-basiert
- Aber ab demnächst auch: **SuperMUC-NG**: “konventionelles” Multicore Design mit ca. 26,7 Pflops/s peak

Motivation: Anwendungsentwicklung bei Komplexität und Heterogenität



- CPU ISA
- Speicher-Hierarchie
  - L1/L2/L3 caches
  - RAM, NUMA
  - NV Memory
  - Disk/SSD/Tape/…
- Interconnect
- Beschleunigerhardware
- …

Ziel: Abstraktion der Heterogenität und Komplexität



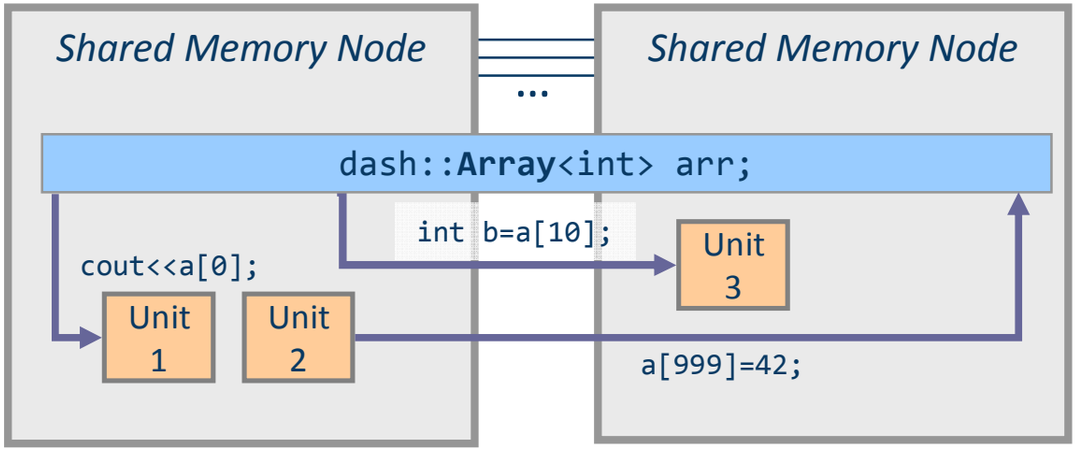
- Höhere Portabilität
- Wartbarkeit und Weiterentwickelbarkeit
- Skalierbarkeit

MEPHISTO

- C++ Metaprogrammierung für heterogene verteilte Systeme
- Keine vollständige Neuentwicklung, sondern:
- **Kombination** und **Erweiterung** von bereits existierenden Komponenten der beteiligten Institutionen
  - Inter-node: **DASH**: Verteilte Datenstrukturen, Parallele Algorithmen
  - Intra-node: **Alpaka**: Abstraktion für parallele Rechenkernel auf Beschleunigerhardware

- Partner
  - Technische Universität Dresden (TUD)
  - Ludwig–Maximilians–Universität (LMU) München
  - Helmholtz–Zentrum Dresden – Rossendorf (HZDR)

Inter-node Abstraktion: DASH



Multiple nodes connected by a high-speed network

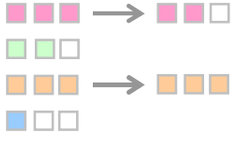
Multiple threads ("units") access **logically** shared memory

- **DASH**:
  - C++ Template Bibliothek von verteilten Datenstrukturen
  - Verallgemeinerung von Shared Memory Programmierung (PGAS)
  - Datenverteilung konfigurierbar
  - Einseitiges Kommunikationsparadigma (Put/Get)

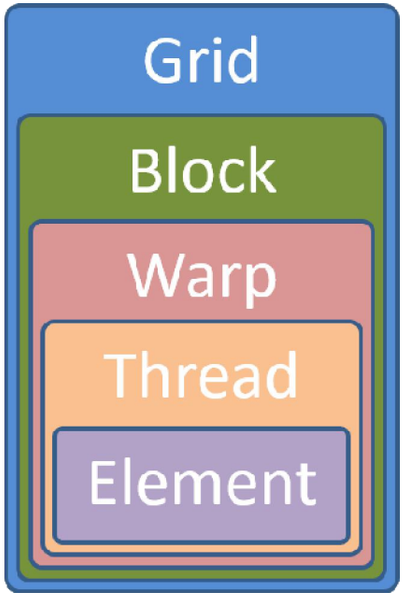
Array<T>

NArray<T, N>

List<T>, **Map**<T>



Intra-node Abstraktion: Alpaka



- **Alpaka**
  - C++ Bibliothek zur Abstraktion von Rechenkerneln auf Knotenebene
  - Entwickelt vom Helmholtz–Zentrum Dresden – Rossendorf (HZDR)
- Mehrere Backends verfügbar
  - CPU (seriell)
  - OpenMP, Intel Threading Building Blocks
  - CUDA, OpenACC

Gemeinsame Lösung: MEPHISTO

DASH

Array<T>

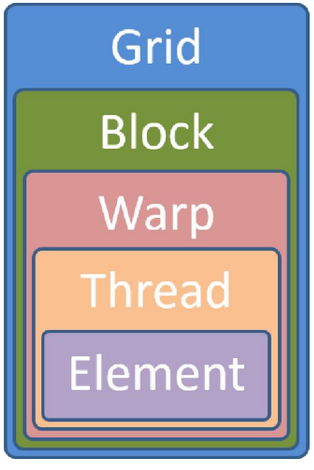
NArray<T, N>

List<T>,

Map<T>

- **Memory Spaces**
  - Möglichkeit DASH Datenstrukturen in konfigurierbaren Speicherbereichen zu allozieren, insbes. GPU Speicher
- **Parallele Algorithmen**
  - “Executors” zur Anbindung von Alpaka Kerneln in DASH
- **Selektive knotenlokale Replikation**
  - “Local Mirror” zur Abbildung globaler Datenstrukturen im lokalen Speicher
- **Abstraktion des Speicherlayouts und –zugriffs**
  - LLAMA

Alpaka



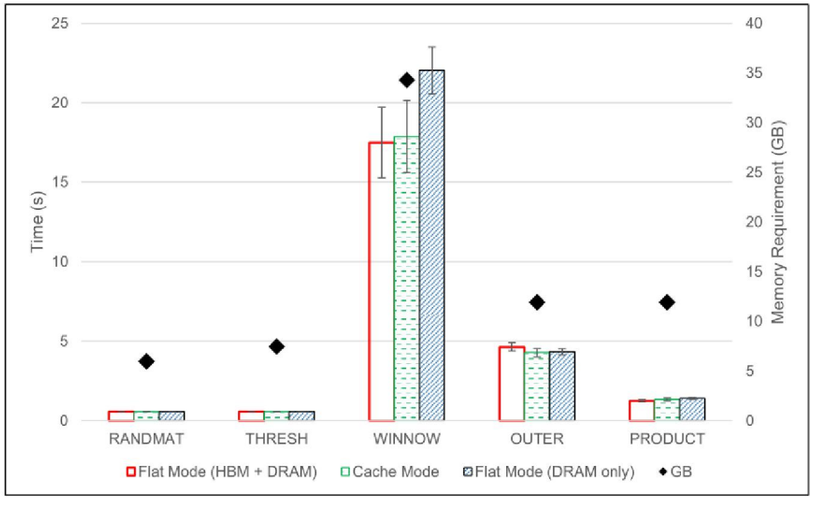
Memory Spaces

- Abstrakte Repräsentation der verschiedenen Speicherorte für Datenelemente sowie deren Eigenschaften
  - Global, Lokal, Beschleuniger (GPU), NUMA Domäne
  - Traits: HighBandwidth, Low Latency, Non-Volatile

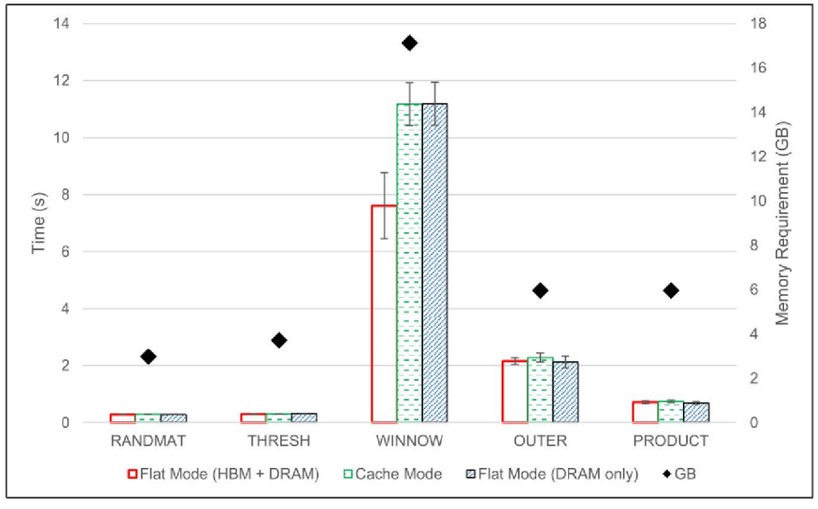
```
template <typename Domain>
class MemorySpace {
 public:
  /// Memory Traits
  using domain_category = Domain;

  //Other Memory Capabilities…
 public:
  void_pointer allocate(...);
  void deallocate(...);
  ...
}


dash::Matrix<double> A{size * size};
dash::Array<double, dash::HBWSpace>  buf{size};
dash::copy(A.blocks(b).begin(),
           A.blocks(b).end(),
           buf.lbegin());
```

Beispiel Memory Spaces

- Fünf Benchmarks aus der “Cowichan” Benchmark suite
  - Platform: Intel Xeon Phi Knights Landing (KNL), 4 bzw. 8 Knoten
  - Manuelle Datenreplikation im HBM kann sigifikante Vorteile haben



(a) $80k \times 80k$ matrix, 4 nodes



(b) $80k \times 80k$ matrix, 8 nodes

R. Kowalewski, T. Fuchs, and K. Fürlinger. **Utilizing Heterogeneous Memory Hierarchies in the PGAS Model**. In Proceedings of the 26th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP 2018), pages 353-357. Cambridge, UK, March 2018.

Executors (1)

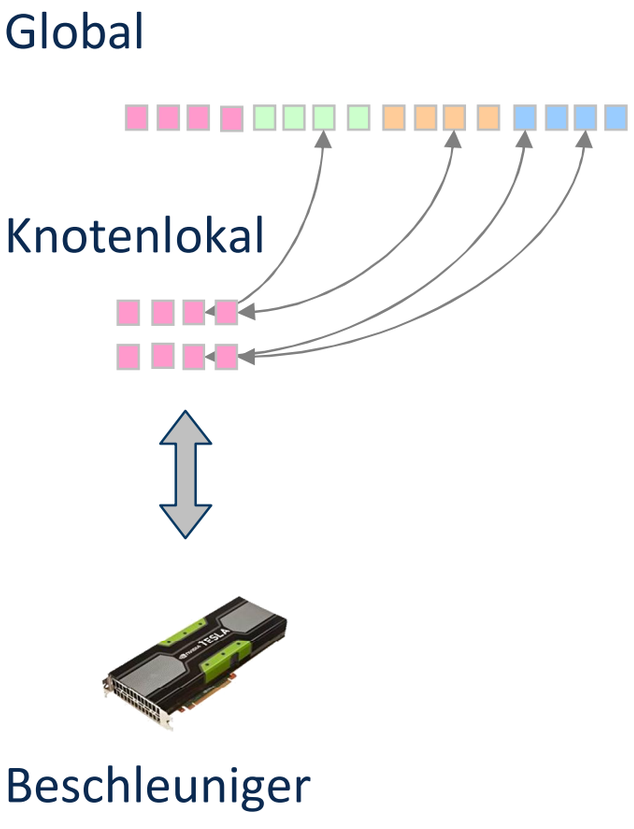
- DASH bietet eine Reihe von parallelen Algorithmen
- Beispiel: dash::transform(begin, end, out, func);
  - Bilde [begin, end) auf [out, …) ab, unter Anwendung von func()
  - Equivalent zu std::transform(…)
- Parallelismus und STL Algorithmen:
  - C++17 führt “Execution Policies” ein
  - std::transform(**parallel_policy**, begin, end, out, func);
  - Ermöglicht parallele und vektorisierte Ausführung der Algorithmen
  - Aber: für MEPHISTO mehr Kontrolle nötig: Welche Beschleunigerkarte?, Speichermanagement, …
- Deswegen: “**Executors**” als Weiterentwicklung der Execution Policies

Executors (2)

- Executors: Noch nicht im C++ Standard, aber mehrere Vorschläge existieren
- Idee:
  - Executors definieren **wie**, **wo**, und **wann** Berechnung geschieht
  - Speichermanagement konfigurierbar
  - Werden vom Programmierer konfiguriert und an den Algorithmus übergeben

```
// user code
mephisto::transform(alpaka_executor, begin, …);


// inside mephisto::transform()
auto result_future = executor().twoway_bulk_execute(
[] (int idx, Result result&) {
  result[idx] = func(first[idx]);
});
```

Local Mirror – Managed Data Replication



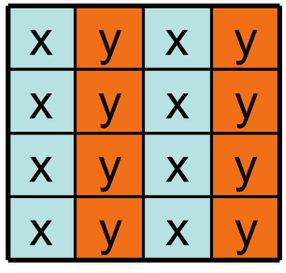
- Idee: lokale Replikation von Teilen der globalen Datenstruktur
  - Ggf. Layouttransformation (cf. LLAMA)
  - Intendierte Verwendung der Lokalen Replika can im Programmiermodell direkt unterstützt werden:
  - zB: Read-Only, Write-Only, Elementweise exklusiver Zugriff, …

LLAMA: Konfigurierbares Speicherlayout

- Datenstrukturen werden aus Benutzer (Programmier-) -sicht abstrakt deklariert

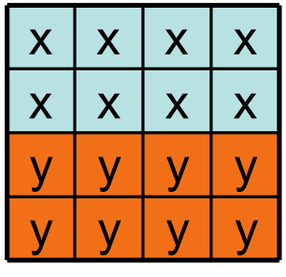
```
struct {
  float x, y;
} pos[8];
```

User code

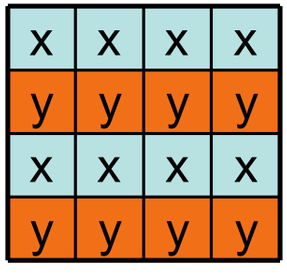


User view ≙ memory layout

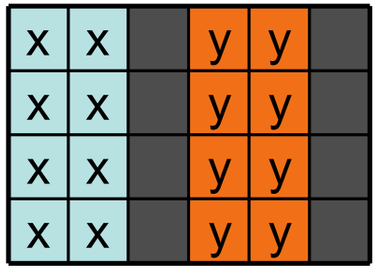
- LLAMA kann das zugrundeliegende Speicherlayout automatisiert anpassen



Struct of Array



Blocking(4)



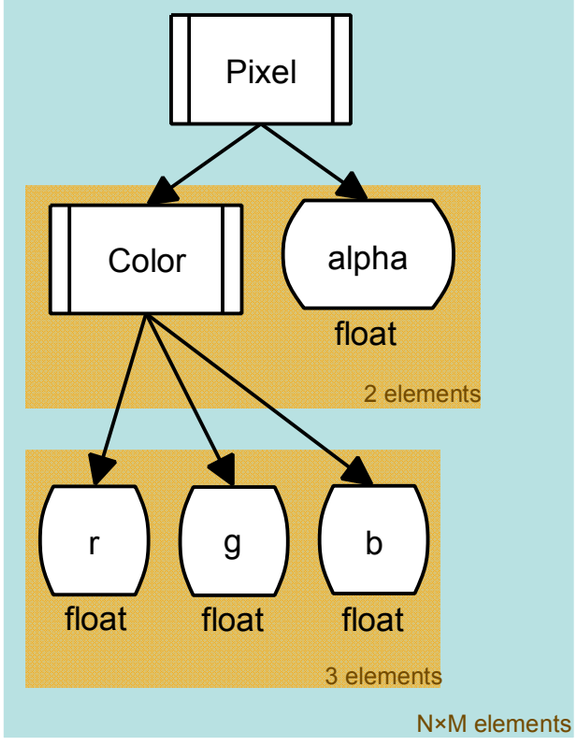
Padding(3)

LLAMA Domains

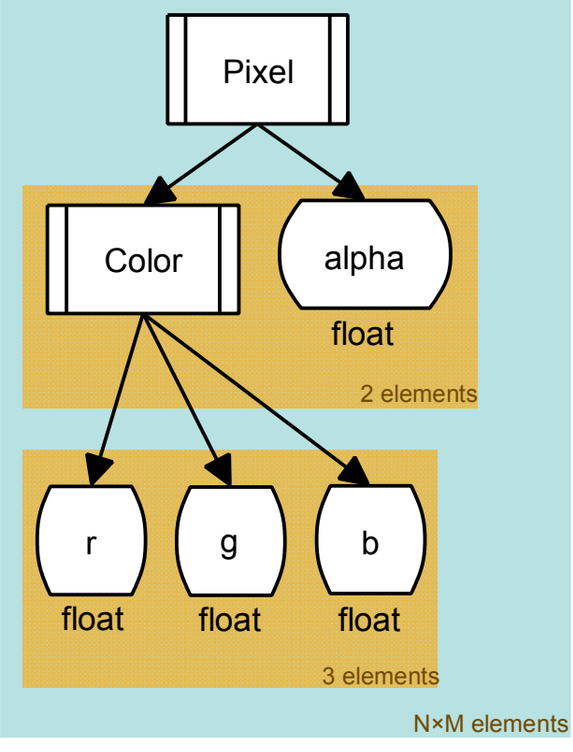
- **User Domain** (UD) ist ein N-dimensionales Array mit Größe N_i in Dimension i
- **Data Domain** (DD) ist eine Baumstruktur:
  - Struct Knoten – Kinder durch Namen oder Index adressierbar
  - Array Knoten – Kinder durch Index adressierbar
  - Skalare – keine Kinder

- Abbildung auf **Byte Domain** und Speicher
  - UD × DD → BD → Speicher

```
struct Pixel {
  struct {
    float r, g, b; } color;
  float alpha; }
image[N][M];
```



```
struct Pixel {
  struct {
    float r, g, b; } color;
  float alpha; }
image[N][M];
```




```
struct r{}, g{}, b{};
struct color{}, alpha{};
using namespace llama;
using Pixel = DS<
  DE<color,
     DS< DE<r,float>,DE<g,float>,DE<b,float>>>,
  DE<alpha,float>
>;

using UD = UserDomain<2>;

// change this line to apply a different layout
using SoA = mapping::SoA<UD,Pixel>;
```

Tags werden zur Benennung benutzt

DS := struct
DE := element

"Struct of Array" Abbildung

LLAMA Example: Image Manipulation

```
const UD userDomain{1000, 1000};

const SoA soa(userDomain);

auto view =
    Factory< SoA >::allocView(mapping);

auto p1 = view(0,0);
p1(color(),r()) = 0.0f; // g(), b(), …
```

- User-Domain mit 1 Mio Elementen
- Struct of Array Abbildung
- Speicherallokation mit Default Konstruktor
- Zugriff auf Pixel (0,0)
- Update Farben

Status und weitere Schritte

- Executor Konzept zur Anbindung von Alpaka Kerneln in DASH prototypisch realisiert
- Demonstrator MEPHISTO Anwendungen in Entwicklung
  - N-body Simulation
  - Bildbearbeitung
- LLAMA und Local Mirror in Entwicklung
  - LLAMA als separate Bibliothek geplant

https://mephisto-hpc.de