max planck institut informatik

The Best of Both Worlds: Flexible Data Structures for Heterogeneous Computing

Robert Strzodka
Integrative Scientific Computing
Max Planck Institut Informatik

My GPU Programming Challenges

max planck institut informatik

- 2000: Conjugate gradient solvers for sparse Ax=b
  - Restrictions of fixed function pipeline in 8bit
- 2003: Multigrid, multi-scale, adaptive time-stepping
  - Restrictions in shaders and data management
- 2005: Glift, random-access GPU data structures
  - Restrictions in use of generic data types in Cg/OpenGL
- 2007: Large-scale simulations on GPU clusters
  - Restrictions of single precision and GPU-CPU communication
- 2009: Preconditioners for ill-conditioned problems
  - Restrictions in debugging of complex code (index tornado)
- Today: Almost full C++ support, debugger&profiler tools
- **Restrictions of C++**

Vector-Valued Data

max planck institut informatik

- Vector-valued data is ubiquitous
  - Class 1: mathematical properties, e.g. multiple derivatives or moments
  - Class 2: discrete features, e.g. colors of a pixel, multiple scores
  - Class 3: per-dimension properties, e.g. coordinates, velocities on a 3D grid

AoS and SoA

max planck institut informatik

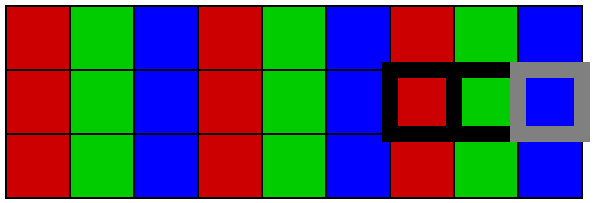
Array of Structs (AoS)

```
struct NormalStruct {
   Type1 comp1;
   Type2 comp2;
   Type3 comp3;
};

typedef NormalStruct
   AoSContainer[SIZE];

AoSContainer container;

container[5].comp3++;
```

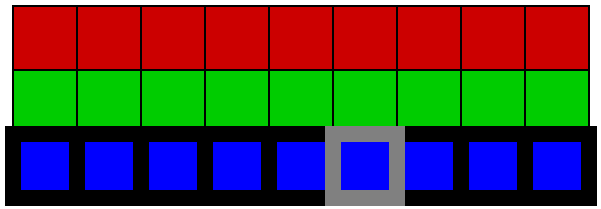


Struct of Arrays (SoA)

```
struct SoAContainer {
   Type1 comp1[SIZE];
   Type2 comp2[SIZE];
   Type3 comp3[SIZE];
};

SoAContainer container;

container.comp3[5]++;
```



Standard C++ Solution

max planck institut informatik

- AoS: container[index].component;
  SoA: container.component[index];
  C++: container.component(index);

- ```
  template <bool isAoS> class CppContainer {};
  template <> class CppContainer<true> {
  public:
    const Type1& comp1(int index) const;  // const
          Type1& comp1(int index);        // mutable
    const Type2& comp2(int index) const;
          Type2& comp2(int index);
    const Type3& comp3(int index) const;
          Type3& comp3(int index);
  private:
    AoSContainer container;
  };
  template <> class CppContainer<false> { … };
  ```

- ```
  CppContainer<true>  aos;    aos.comp3(5)= 0.123;
  CppContainer<false> soa;    soa.comp3(5)= 0.123;
  ```

Operating on Container Elements

max planck institut informatik

- ```
  struct NormalStruct {
    Type1 comp1;
    Type2 comp2;
    Type3 comp3;
  };
  typedef NormalStruct  AoSContainer[SIZE];
  ```

- ```
  NormalStruct single;
  AoSContainer container;
  ```

- In-place update of single and indexed structs

  ```
  void update( NormalStruct& s ) { s.comp3 += s.comp1; }
  ```

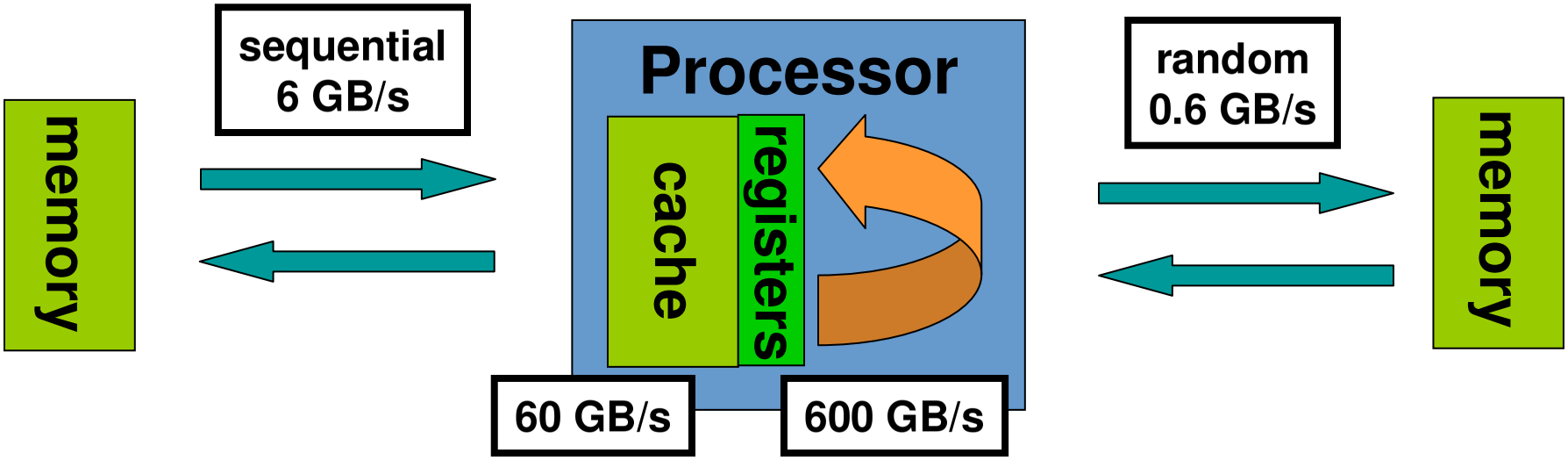
- ```
  update( single );                  // OK
  update( container[5] );            // OK
  ```

- This is not possible with standard SoA/C++ syntax:

  ```
  container.comp3(5);
  ```
  ```

Typical Bandwidth Relations

max planck institut informatik

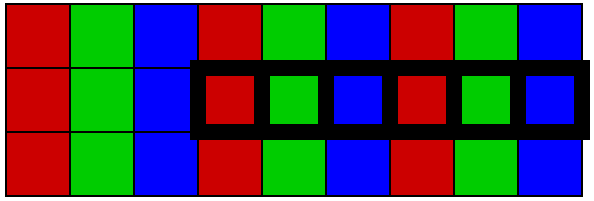


Parallel Access in AoS and SoA

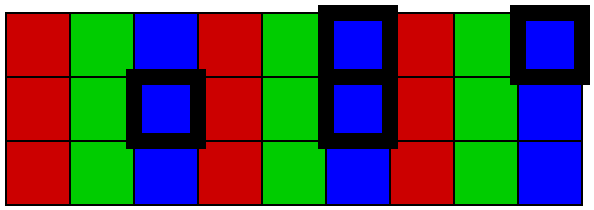
max planck institut informatik

Array of Structs (AoS)

```
container[4].comp{1,2,3}
container[5].comp{1,2,3}
```

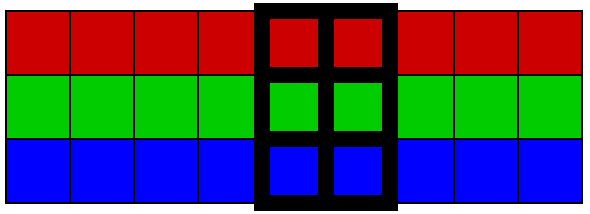


```
container[1].comp3
container[2].comp3
container[3].comp3
container[4].comp3
```

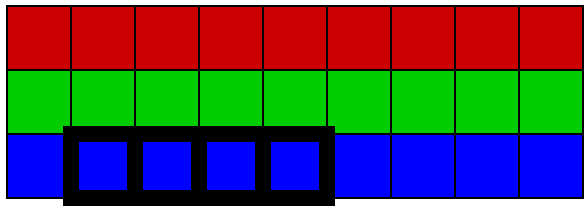


Struct of Arrays (SoA)

```
container.comp{1,2,3}[4]
container.comp{1,2,3}[5]
```



```
container.comp3[1]
container.comp3[2]
container.comp3[3]
container.comp3[4]
```



Goals

max planck institut informatik

- A one-parameter switch between AoS and SoA layout
- The same access syntax independent of the data layout: `container[index].component`
- The same functions to operate on single and indexed structs: `update( container[5] );`
- Minimal changes to existing AoS code

AoS + SoA = ASA

max planck institut informatik

Array of Structs (AoS)

```
struct NormalStruct {
   Type1 comp1;
   Type2 comp2;
   Type3 comp3;
};

typedef NormalStruct
   Container[SIZE];

NormalStruct single
   Container container;

void
   update(NormalStruct& s);

container[index].comp3++;
update( container[5] );
```

Array of Structs of Arrays (ASA)

```
template <ID t_id=ID_value>
struct FlexibleStruct {
   typedef ASAGroup<Type1,t_id> ASX_ASA;
   union{ Type1 comp1; ASX_ASA d1; };
   union{ Type2 comp2; ASX_ASA d2; };
   union{ Type3 comp3; ASX_ASA d3; };
};
typedef ASX::Array<FlexibleStruct,
   SIZE, ??? > Container;

FlexibleStruct<> single
   Container container;

template <ID t_id> void
   update(FlexibleStruct<t_id>& s);

container[index].comp3++;
update( container[5] );

??? = ASX::AOS or ASX::SOA
```

ASA Syntax

max planck institut informatik

- ```
  template <ID t_id=ID_value> struct FlexibleStruct {
    typedef ASAGroup<Type1,t_id> ASX_ASA;
    union{ Type1 comp1; ASX_ASA d1; };
    union{ Type2 comp2; ASX_ASA d2; };
    union{ Type3 comp3; ASX_ASA d3; };
  };
  ```
- Assumes: `sizeof(Type1) >= {sizeof(Type2), sizeof(Type3)}`
- ```
  typedef ASAGroup<GroupType,t_id> ASX_ASA;
  sizeof(GroupType) >= sizeof(Type{1,2,3})
  ```
- ```
  template <ID t_id=ID_value> struct FlexibleStruct {
    typedef ASAGroup<Type1,t_id> ASX_ASA;
    union{ Type1 comp1;                              ASX_ASA d1; };
    union{ struct{Type2 comp2; Type3 comp3;}; ASX_ASA d2; };
  };
  ```
- Assumes: `sizeof(Type1) >= sizeof(Type2) + sizeof(Type3)`
- Anonymous `union` is part of C++, anonymous `struct` supported by large
  ```

References

max planck institut informatik

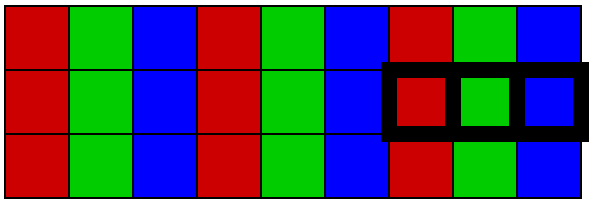
Array of Structs (AoS)

```
void
    update(NormalStruct& s);

update( single );
update( container[5] );

NormalStruct&
   ref= container[5];

ref.comp3= 0.123;
```



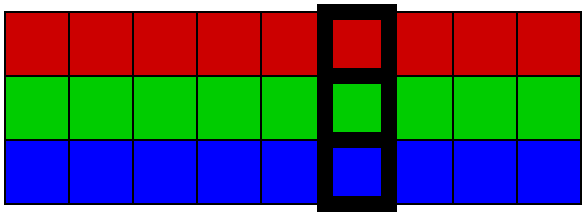
Array of Structs of Arrays (ASA)

```
template <ID t_id> void
   update(FlexibleStruct<t_id>& s);

update( single );
update( container[5] );

Container::reference
   ref= container[5];

ref.comp3= 0.123;
```



More Details

max planck institut informatik

- Paper
  - GPU Computing Gems
  - Abstraction for AoS and SoA layout
- Code
  - ASX library with CPU and GPU examples
  - Soon at www.mpii.de/~strzodka

Pros and Cons

max planck institut informatik

- Algorithm
  - Algorithms are independent of the data layout
  - Above holds only if anonymous `struct` is supported
- Coding
  - Only definitions (container, references) change
  - Container `SIZE` must be a compile-time constant
- Optimization
  - Performance of AoS vs. SoA can be evaluated rapidly
  - Component groups are hard-coded by the `unions{}`

Future

max planck institut informatik

- Adapting Abstractions
  - New language (Compiler ☺, User ☺, Acceptance ☹)
  - Language extension (Compiler ☺, User ☺, Acceptance 😐)
  - Library+Code pattern (Compiler 😐, User 😐, Acceptance ☺)

- GPU Programming Challenges
  - Previously: Catching up with the CPU standard
  - Today: Evolving general parallel abstractions

  - Impulse: New Abstractions more important than full C++
  - Impulse: Higher extension acceptance benefits GPUs

max planck institut informatik

The Best of Both Worlds: Flexible Data Structures for Heterogeneous Computing

Robert Strzodka
Integrative Scientific Computing
Max Planck Institut Informatik