A Novel Data Layout Optimization in BiSheng Compiler

Ehsan Amiri, Hao Jin, Mehrnoosh Heidarpour, Bryan Chan, Nigel Yu

www.huawei.com

HUAWEI TECHNOLOGIES CO., LTD.

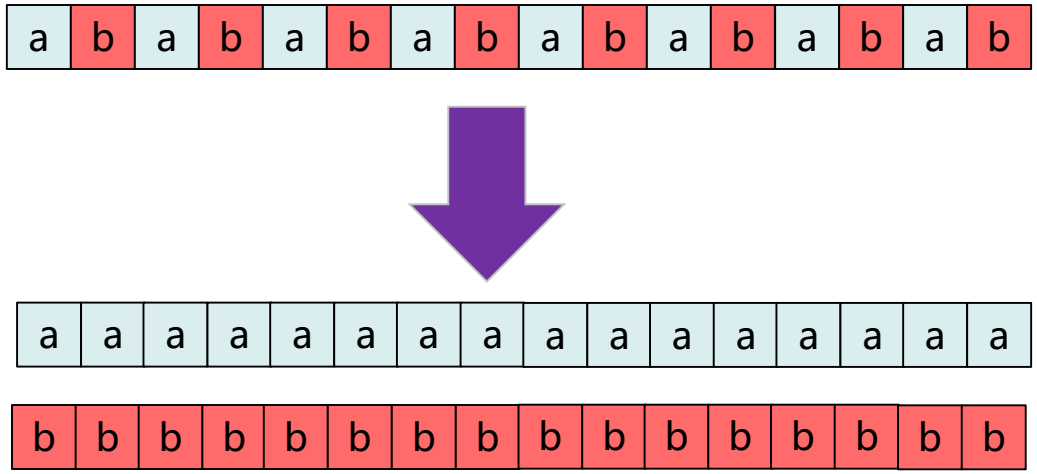
HUAWEI

BiSheng Compiler

- **LLVM-based C/C++/Fortran compiler developed by Huawei.**
  - Under development since late 2019.
- **Primarily optimized for Kunpeng AArch64 servers.**
- **Supports X86 as well.**
- **We have talked about BiSheng internals in LLVM Dev meeting and other conferences in the past couple of years.**

Data layout optimizations

- **Many well-known data layout optimizations in the literature.**
  - Array of Structure to Structure of Arrays. (AKA structure peeling).
    - See the picture.
  - Array reorganization.
  - Structure repacking.



- **Improve spatial locality of memory accesses**
  - Better cache utilization.
  - Better memory bandwidth utilization.

- **Critical for SPEC CPU benchmarks.**

- **Previous talk at LATHC workshop (CGO 2023) covers our new approach to structure peeling**

The new data layout optimization

- **We introduce our new optimization: Nested Container Flattening**
- **Will explain the name during this presentation.**

- **What is covered:**
  - High level description of the transformation and the motivation behind it.
  - Major issues in legality analysis

- **What is not covered**
  - Details of transformation
  - Details of legality analysis
  - Cost analysis

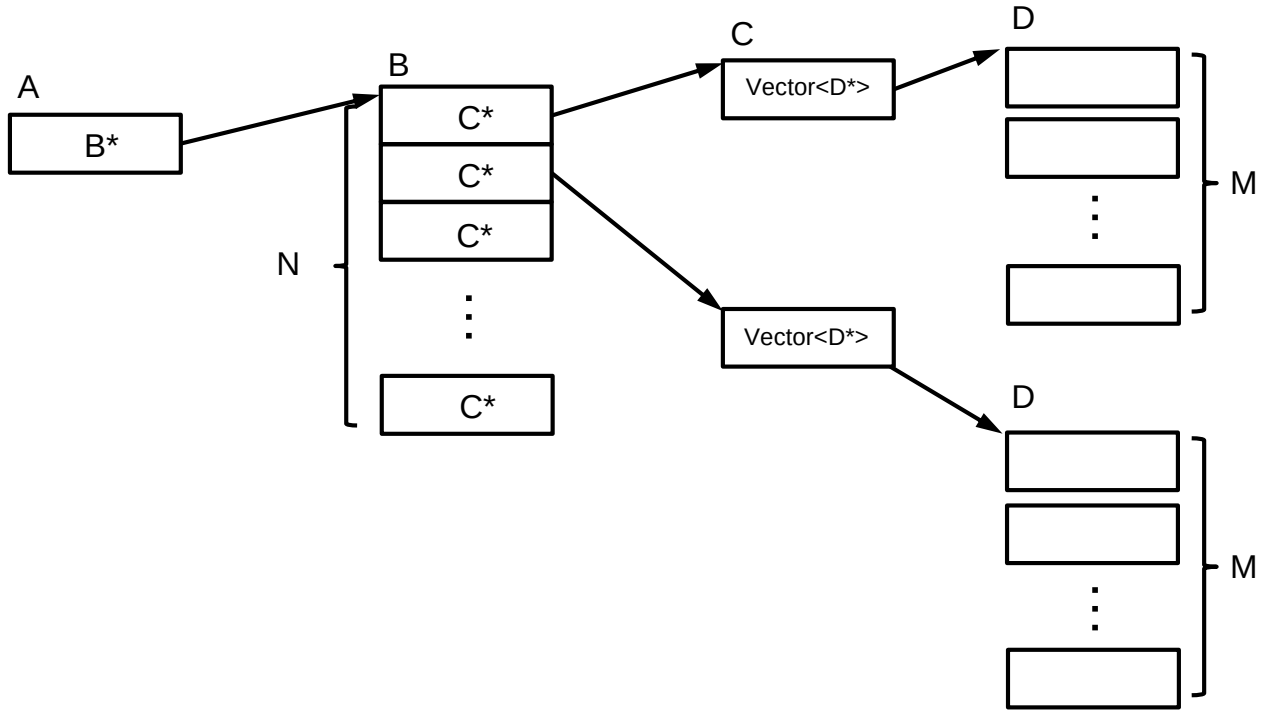
The problem

```
class A {

public:
   B *b;

};
```

```
class B {

public:
   C *c;

};
```

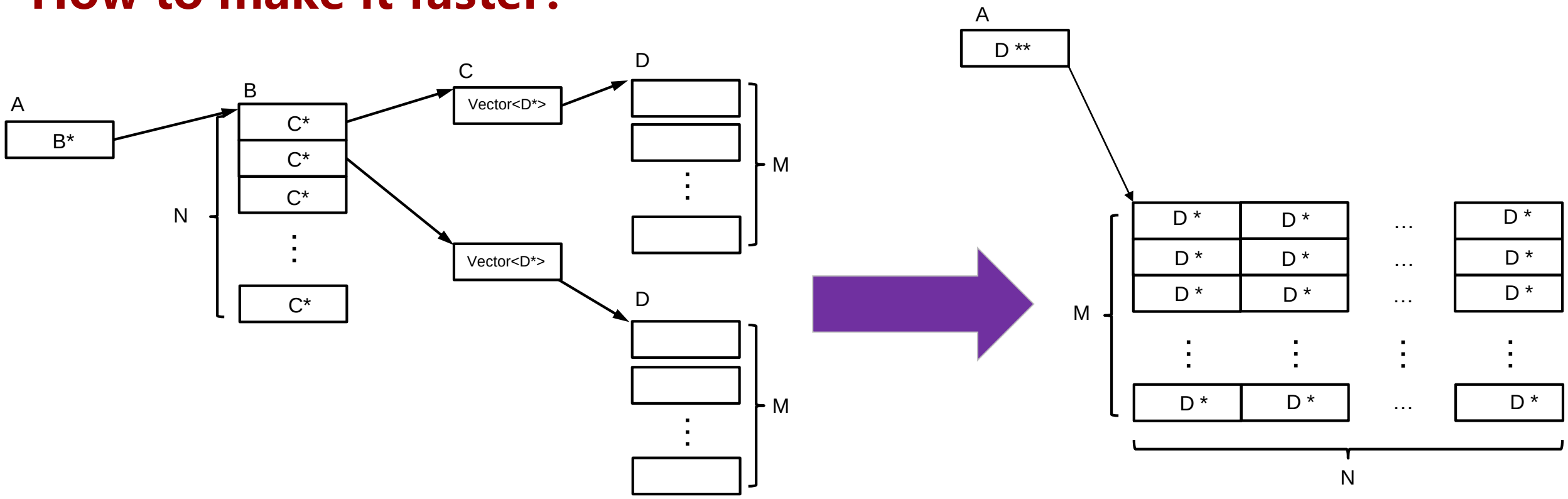
```
class C {

public:
   std::vector<D *> d;

};
```

```
void func(A *a) {
  ……
  a->b[i].c->d[k]->DoWork();
  ……
}
```



- Objects in different arrays are likely to be in different memory locations.
- Each load is likely to cause a cache miss.
- Intermediate loads has no other use.

How to make it faster?



- All D pointers reachable from an A object are now copied within that object.
- Old D pointers may or may not be removed.
- Nested Container Flattening

Legality analysis

- **Compiler needs very deep understanding of life time of the objects involved.**
- **This requires the ability to analyze where the objects are created and where the fields are written to or read from.**
- **It looks like we need pointer analysis.**
- **In a C++ workload, it is not likely that pointer analysis can reach the precision level that we need, while being fast enough.**
- **We mostly rely on preserving C++ source code level information and analyzing that in the mid-end.**
  - › Full legality analysis is complex and has many details.
  - › We just explain two central concepts: Nested objects, Containers.
  - › This completes explanation of the name: Nested Container Flattening.

Legality analysis: Nested Objects

- **Saying b is *Nested* in class A means**
  - There is no access to b outside of class A member functions.
  - Entire lifetime of b, from allocation to deallocation, is within the lifetime of an A object.
  - Relying on C++ source code information to prove this property requires considering many different C++ features and intricacies. Some pointer analysis is still needed.
  - It is easier to reason about a nested B object.
    - e.g. Consider only relevant B member functions.
    - See the toy example on the right.
    - Only two functions are relevant.

```
class B {
  int x;
  int y;

public:
  void func1();
  void func2();
  void func3();
  void func4();
};
```

```
class A {

  B *b;

public:
  void func() {
    b = new B();
    b->func1();
    b->func2();
    delete b;
  }
};
```

Legality Analysis: Containers

- What does it mean to say object b is a container?
- Intuitively, that means some information is passed to b and preserved.

```
class A {

   B *b; // b is nested in A.
   // only b->addNewObject() is called.


};
```

```
class B {
  int x;
  C* c; // array of C objects

public:
  void addNewObject(C &c);
  void removeLast();
  void updateLast();
  void updateAll();
};
```

How useful is an optimization like this?

- **Data layout optimizations are usually too complex.**
  - It is hard to prove their legality
  - It is hard to perform the transformation
- **The optimization is not triggered that often.**
- **We get 35% improvement on a SPEC benchmark, what else?**
- **The infrastructure developed is usually useful beyond the optimization itself.**
- **Many DL opts require a good pointer analysis.**
  - Pointer analysis is useful beyond these optimizations.
- **We believe our ideas for legality analysis of this new transformation are useful beyond this optimization.**
  - Some potential applications of these techniques for other optimizations are under investigation.

Thank you

www.huawei.com

Copyright©2016 Huawei Technologies Co., Ltd. All Rights Reserved.
The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.