CoDiPack 2.0: Lessons learned and new ideas

M. Sagebaum, J. Blühdorn

AG Scientific Computing
TU Kaiserslautern

23rd European Workshop on Automatic Differentiation

Overview

- History
- Lessons learned
- New ideas
- Programming performance mistakes
- Current performance values

History of CoDiPack

- **19.06.2015**: Initial release v1.0
  - Jacobian taping approach with linear index management
  - External function support
- **23.12.2015**: Release v1.1
  - Jacobian taping approach with reuse index management
  - Preaccumulation support
  - Template files for binary and unary operators
- **07.04.2016**: Release v1.2
  - `ReferenceActiveReal`
  - Reuse index management with assign optimization
  - Full vector support
  - Module based implementation of tapes
- **14.09.2016**: Release v1.3
  - Primal value taping approach
- **16.02.2017**: Release v1.4
  - Higher order derivative helper
  - Read/write tapes from files
  - Management of multiple tapes
- **18.12.2017**: Release v1.5
  - Custom adjoint vector support
  - `TapeVectorHelper`
  - `ExternalFunctionHelper`
  - `PreaccumulationHelper`
  - `StatementPushHelper`
- **15.03.2018**: Release v1.6
  - Forward evaluation of tapes (basic support)
  - Generalized function for stack evaluation
- **30.10.2018**: Release v1.7
  - Forward evaluation of tapes (full support)
  - Primal evaluation of tapes
- **07.01.2019**: Release v1.8
  - Automatic Jacobian combination for statements
  - Dropped tape modules approach, switched to inheritance of structures
- **10.10.2019**: Release v1.9
  - `EvaluationHelper`
  - `TapeHelper`
  - Expression logic definition rework

History of CoDiPack

- **19.06.2015**: Initial release v1.0
  - Jacobian taping approach with linear index management
  - External function support
- **23.12.2015**: Release v1.1
  - Jacobian taping approach with reuse index management
  - Preaccumulation support
  - Template files for binary and unary operators
- **07.04.2016**: Release v1.2
  - `ReferenceActiveReal`
  - Reuse index management with assign optimization
  - Full vector support
  - Module based implementation of tapes
- **14.09.2016**: Release v1.3
  - Primal value taping approach
- **16.02.2017**: Release v1.4
  - Higher order derivative helper
  - Read/write tapes from files
  - Management of multiple tapes
- **18.12.2017**: Release v1.5
  - Custom adjoint vector support
  - `TapeVectorHelper`
  - `ExternalFunctionHelper`
  - `PreaccumulationHelper`
  - `StatementPushHelper`
- **15.03.2018**: Release v1.6
  - Forward evaluation of tapes (basic support)
  - Generalized function for stack evaluation
- **30.10.2018**: Release v1.7
  - Forward evaluation of tapes (full support)
  - Primal evaluation of tapes
- **07.01.2019**: Release v1.8
  - Automatic Jacobian combination for statements
  - Dropped tape modules approach, switched to inheritance of structures
- **10.10.2019**: Release v1.9
  - `EvaluationHelper`
  - `TapeHelper`
  - Expression logic definition rework

History of CoDiPack

- **19.06.2015**: Initial release v1.0
  - Jacobian taping approach with linear index management
  - External function support
- **23.12.2015**: Release v1.1
  - Jacobian taping approach with reuse index management
  - Preaccumulation support
  - Template files for binary and unary operators
- **07.04.2016**: Release v1.2
  - `ReferenceActiveReal`
  - Reuse index management with assign optimization
  - Full vector support
  - Module based implementation of tapes
- **14.09.2016**: Release v1.3
  - Primal value taping approach
- **16.02.2017**: Release v1.4
  - Higher order derivative helper
  - Read/write tapes from files
  - Management of multiple tapes
- **18.12.2017**: Release v1.5
  - Custom adjoint vector support
  - `TapeVectorHelper`
  - `ExternalFunctionHelper`
  - `PreaccumulationHelper`
  - `StatementPushHelper`
- **15.03.2018**: Release v1.6
  - Forward evaluation of tapes (basic support)
  - Generalized function for stack evaluation
- **30.10.2018**: Release v1.7
  - Forward evaluation of tapes (full support)
  - Primal evaluation of tapes
- **07.01.2019**: Release v1.8
  - Automatic Jacobian combination for statements
  - Dropped tape modules approach, switched to inheritance of structures
- **10.10.2019**: Release v1.9
  - `EvaluationHelper`
  - `TapeHelper`
  - Expression logic definition rework

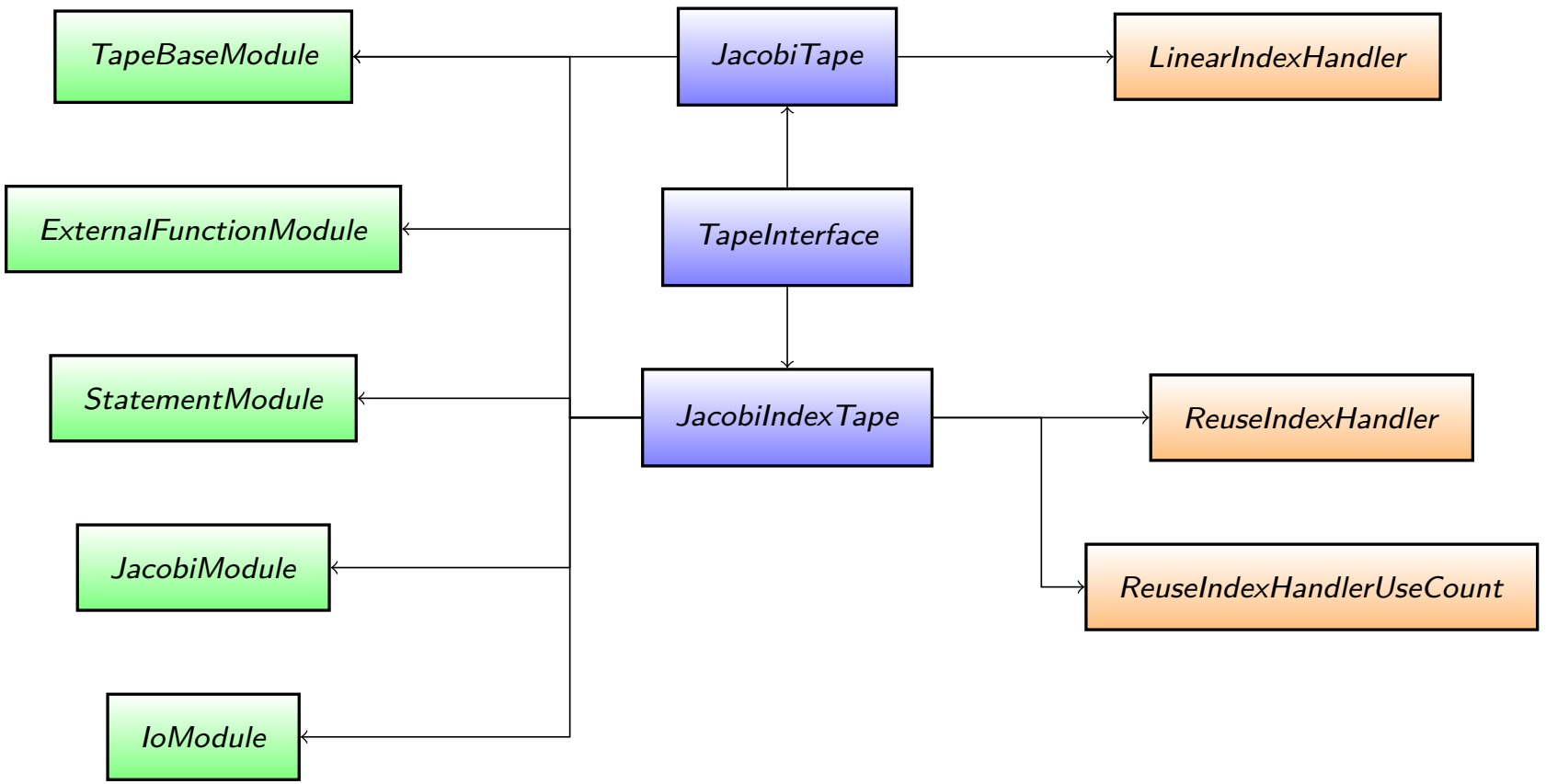
History of CoDiPack

- **19.06.2015**: Initial release v1.0
  - Jacobian taping approach with linear index management
  - External function support
- **23.12.2015**: Release v1.1
  - Jacobian taping approach with reuse index management
  - Preaccumulation support
  - Template files for binary and unary operators
- **07.04.2016**: Release v1.2
  - `ReferenceActiveReal`
  - Reuse index management with assign optimization
  - Full vector support
  - Module based implementation of tapes
- **14.09.2016**: Release v1.3
  - Primal value taping approach
- **16.02.2017**: Release v1.4
  - Higher order derivative helper
  - Read/write tapes from files
  - Management of multiple tapes
- **18.12.2017**: Release v1.5
  - Custom adjoint vector support
  - `TapeVectorHelper`
  - `ExternalFunctionHelper`
  - `PreaccumulationHelper`
  - `StatementPushHelper`
- **15.03.2018**: Release v1.6
  - Forward evaluation of tapes (basic support)
  - Generalized function for stack evaluation
- **30.10.2018**: Release v1.7
  - Forward evaluation of tapes (full support)
  - Primal evaluation of tapes
- **07.01.2019**: Release v1.8
  - Automatic Jacobian combination for statements
  - Dropped tape modules approach, switched to inheritance of structures
- **10.10.2019**: Release v1.9
  - `EvaluationHelper`
  - `TapeHelper`
  - Expression logic definition rework

History of CoDiPack

- **19.06.2015**: Initial release v1.0
  - Jacobian taping approach with linear index management
  - External function support
- **23.12.2015**: Release v1.1
  - Jacobian taping approach with reuse index management
  - Preaccumulation support
  - Template files for binary and unary operators
- **07.04.2016**: Release v1.2
  - `ReferenceActiveReal`
  - Reuse index management with assign optimization
  - Full vector support
  - Module based implementation of tapes
- **14.09.2016**: Release v1.3
  - Primal value taping approach
- **16.02.2017**: Release v1.4
  - Higher order derivative helper
  - Read/write tapes from files
  - Management of multiple tapes
- **18.12.2017**: Release v1.5
  - Custom adjoint vector support
  - `TapeVectorHelper`
  - `ExternalFunctionHelper`
  - `PreaccumulationHelper`
  - `StatementPushHelper`
- **15.03.2018**: Release v1.6
  - Forward evaluation of tapes (basic support)
  - Generalized function for stack evaluation
- **30.10.2018**: Release v1.7
  - Forward evaluation of tapes (full support)
  - Primal evaluation of tapes
- **07.01.2019**: Release v1.8
  - Automatic Jacobian combination for statements
  - Dropped tape modules approach, switched to inheritance of structures
- **10.10.2019**: Release v1.9
  - `EvaluationHelper`
  - `TapeHelper`
  - Expression logic definition rework

CoDiPack tape layout

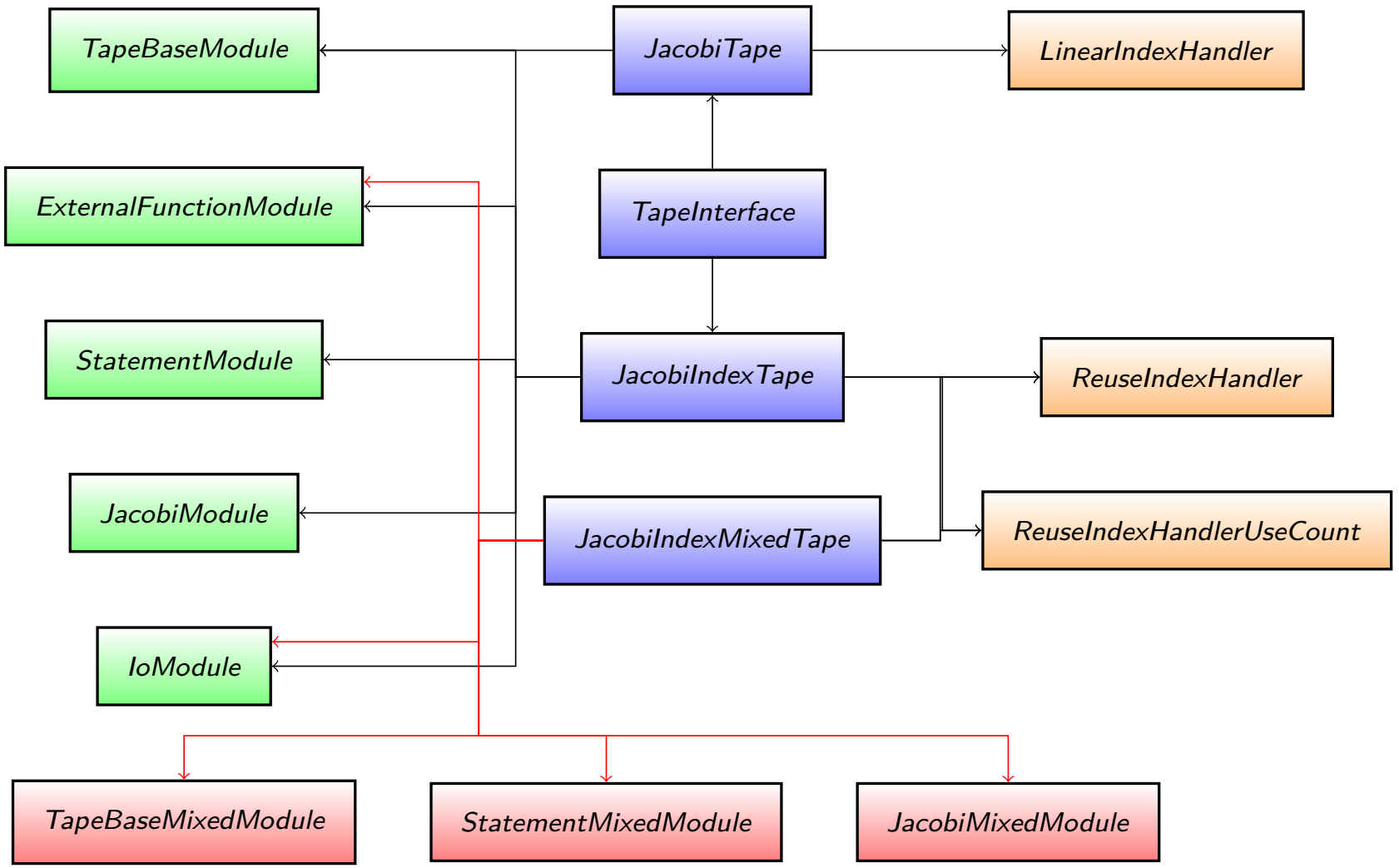


CoDiPack tape layout - Changes for mixed AD types

What do we want to support?

```
AReal r;          // Active Real
AComplex c;       // Active Complex
ASimd s;          // Active Simd

GlobalTape::registerInput(r, c, s);

AComplex w1 = r * c;                // Mixed type computations
ASimd w2 = (norm(c) + r) * s;

GlobalTape::registerOutput(w1, w2);
w1.gradient() = 2.0 + 3.0 * I;
w2.gradient() = {1.0, 1.0, 1.0, 1.0};

GlobalTape::evaluate();
```

CoDiPack tape layout - Changes for mixed AD types



CoDiPack tape layout - Modular approach

Drawbacks:

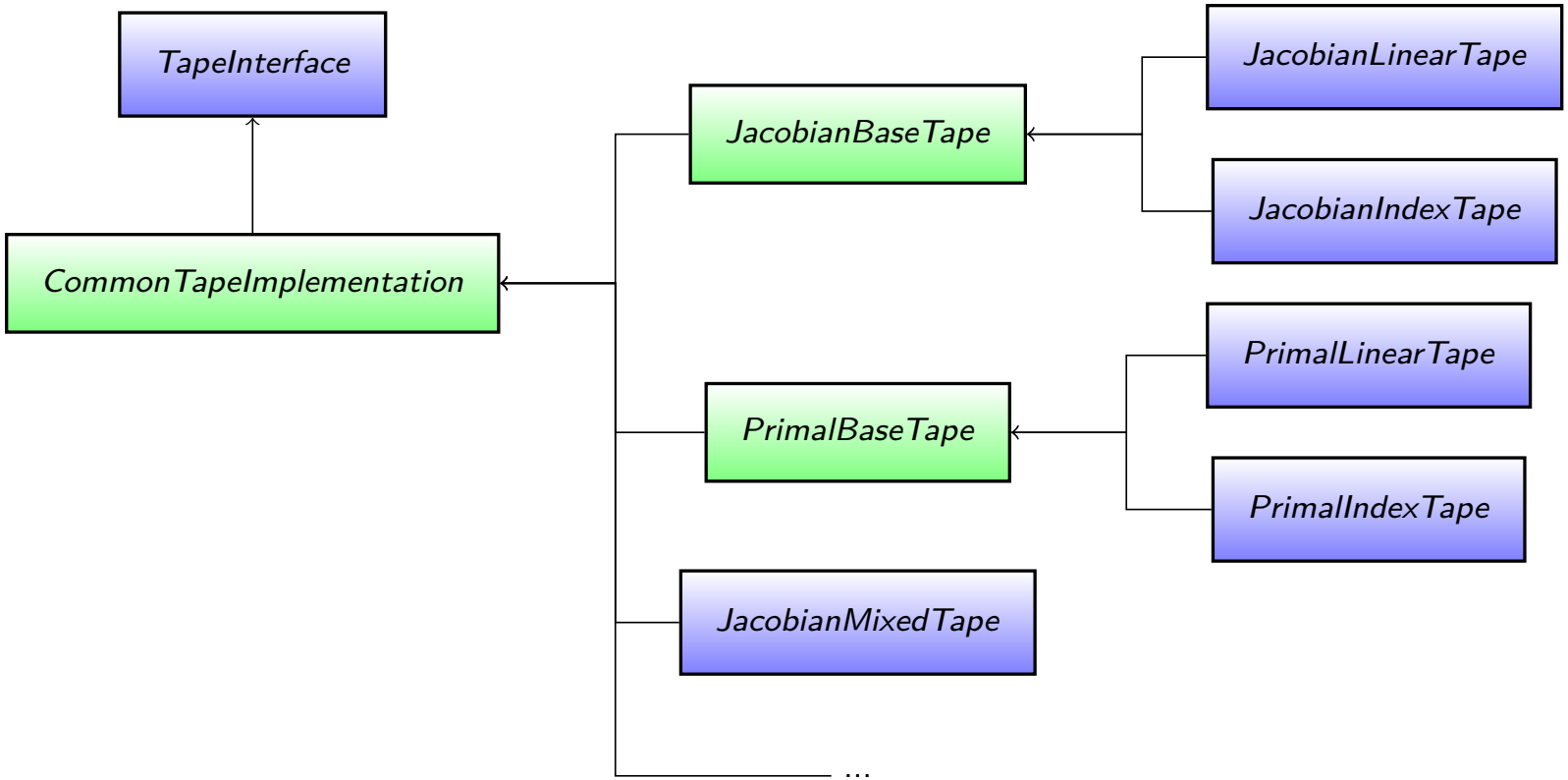
- Difficult to find function implementation (Which module?)
- IDE support (e.g. auto completion) only partly working
- Hard to understand
  - Module A calls module B calls module C calls module A calls module D
- Difficult to generalize
  - Module needs to cover all different usecases
    - Normal, Mixed, Threaded, OpenMP, etc.
  - Mostly contradicting requirements

Advantages:

- No code copy
- Faster
  - w.r.t compile time, evaluation time
  - (Original motivation for modularized approach)

CoDiPack tape layout - CoDiPack 2.0 approach

- Traditional inheritance strategy
- Basic implementation contains only:
  - Forward function
  - Self contained data structures
- Each taping approach has a single implementation file



CoDiPack statement handling

How is a statement stored?

```
w = (a + b) * (c - d);
```

The call hierarchy is:

ActiveReal	→	Tape	→	Expression	→	Tape
operator =	→	store	→	calcGradient	→	pushJacobi

CoDiPack statement handling

How is a statement stored?

```
w = (a + b) * (c - d);
```

The call hierarchy is:

ActiveReal	→	Tape	→	Expression	→	Tape
operator =	→	store	→	calcGradient	→	pushJacobi

- Expression can be:
  - UnaryExpression, BinaryExpression11, BinaryExpression01, BinaryExpression10, ActiveReal, RefActiveReal

  ⇒ 6 different implementation of calcGradient
- Primal value tapes require additional functions:
  - getValueStatic, calcGradientStatic, perValueAction, perConstantAction, pushLazyJacobies

  ⇒ In total 36 different method implementations

CoDiPack statement handling

Drawbacks:

- Code copy (Especially function definitions)
- Difficult to know all `Expression` implementations
- Difficult to add new `Expression` implementation
- Hard to understand (Large call hierarchy)
- Tape specific logic in expression implementation

Advantages:

- No template magic
- Faster
  - w.r.t compile time, evaluation time
  - (Original motivation for deep call hierarchy)

CoDiPack statement handling - CoDiPack 2.0 approach

General idea:

- Make it possible to evaluate custom logic operations on an expression graph

CoDiPack statement handling - CoDiPack 2.0 approach

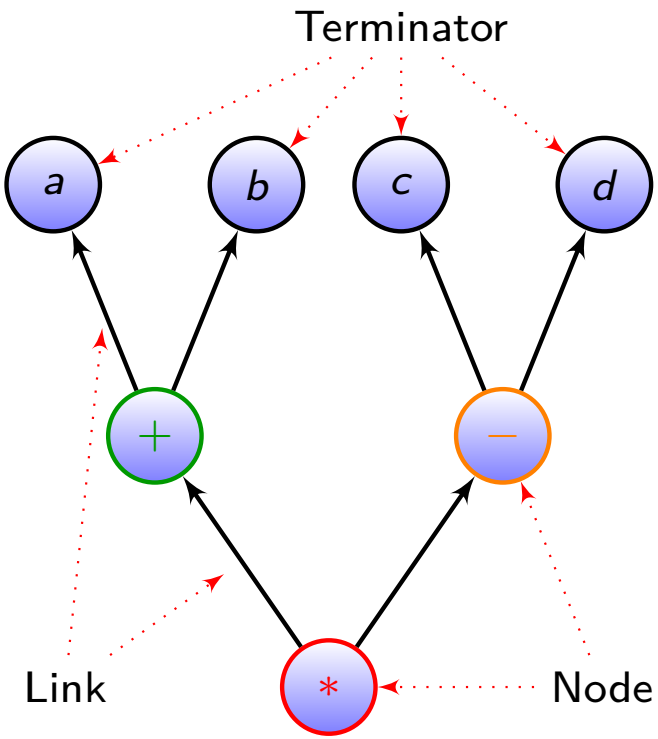
General idea:

- Make it possible to evaluate custom logic operations on an expression graph

Statement:

```
w = (a + b) * (c - d);
```

Graph:



CoDiPack statement handling - CoDiPack 2.0 approach

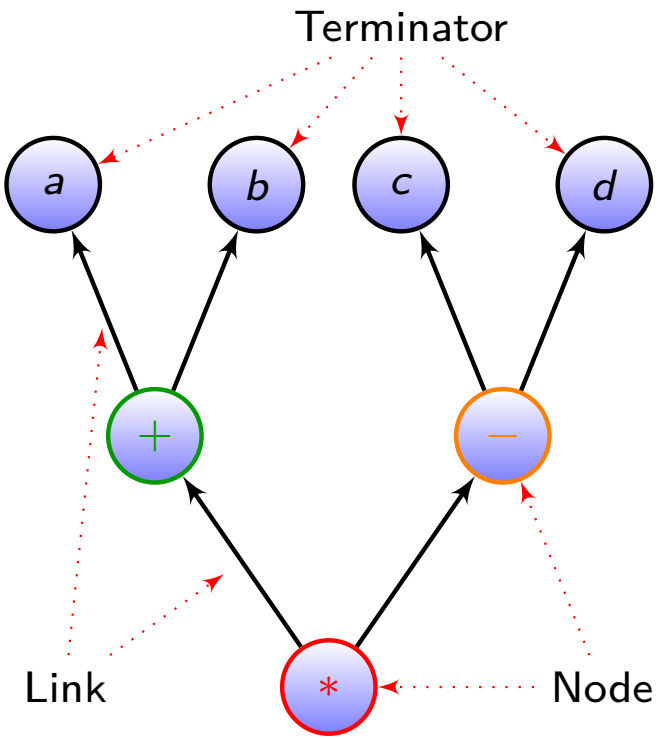
General idea:

- Make it possible to evaluate custom logic operations on an expression graph

Statement:

```
w = (a + b) * (c - d);
```

Graph:



- User needs to be able to declare logic for each:
  - Node
  - Link
  - Termination node

CoDiPack statement handling - CoDiPack 2.0 approach

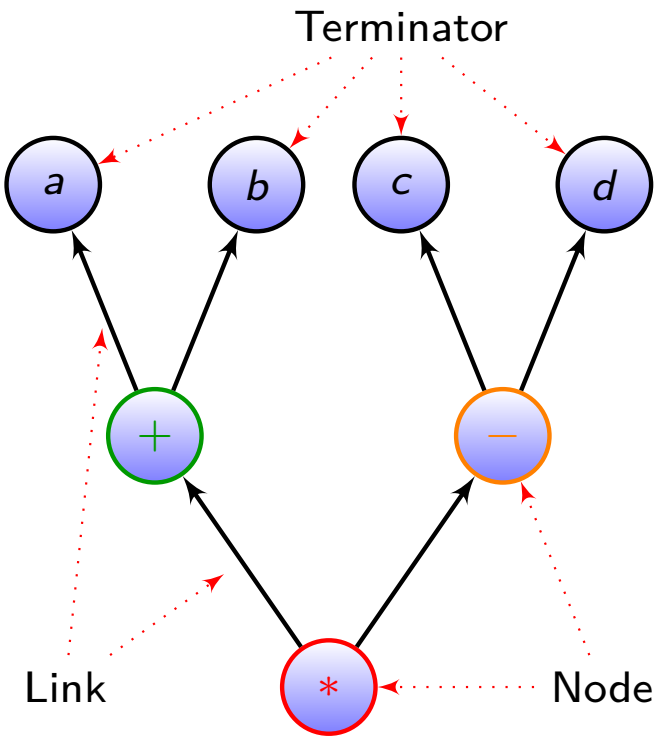
General idea:

- Make it possible to evaluate custom logic operations on an expression graph

Statement:

```
w = (a + b) * (c - d);
```

Graph:



- User needs to be able to declare logic for each:
  - Node
  - Link
  - Termination node
- Implement interface:

```
struct TraversalLogic {
  template<typename Node, typename ... Args>
  void node(Node const& node, Args&& ... args);

  template<typename Node, typename ... Args>
  void term(Node const& node, Args&& ... args);

  template<size_t LeafNumber, typename Leaf,
           typename Root, typename ... Args>
  void link(Leaf const& leaf, Root const& root,
            Args&& ... args);
};
```

CoDiPack statement handling - CoDiPack 2.0 approach

Example: Jacobian computation and storing

```
template<typename Real>
struct JacobianLogic : public TraversalLogicBase {

  template<size_t LeafNumber, typename Leaf, typename Root, typename ... Args>
  void link(Leaf const& leaf, Root const& root, Real const& jacobian, Args&& ... args) {

    Real curJacobian = root.template getJacobian<LeafNumber>() * jacobian;
    toNode(leaf, curJacobian, std::forward<Args>(args)...);
  }

  template<typename Node, typename DataVector>
  enableIfLhsExpression<Node> term(Node const& node, Real jacobian,
                                   DataVector& dataVector) {
    dataVector.pushData(jacobian, node.getIdentifier());
  }
};
```

- `link`: Accesses the Jacobian with respect to this link
- `term`: Stores the computed Jacobian
- `node`: Using default logic (Forward to all links)
- `enableIfLhsExpression`: Only evaluate this for active types

CoDiPack statement handling - CoDiPack 2.0 approach

Drawbacks:

- Lots of template magic
- Difficult to select correct termination pointer (SFINAE principle)
- Lots of steps with debugger

Advantages:

- No code copy
- Implementation local to tape
- Easy to understand

Observations:

- No compile time drawback
- No performance drawback

CoDiPack expression implementation

CoDiPack: 6 classes, 11 functions

- Classes: `UnaryExpression`, `BinaryExpression11`, `BinaryExpression10`, `BinaryExpression01`, `ActiveReal`, `RefActiveReal`
- Functions: `calcGradient()`, `calcGradient(multiplier)`, `derv11`, `devr01`, `derv10`, `derv11M`, `derv01M`, `derv10M`, `gradientA`, `gradientB`, `gradient`

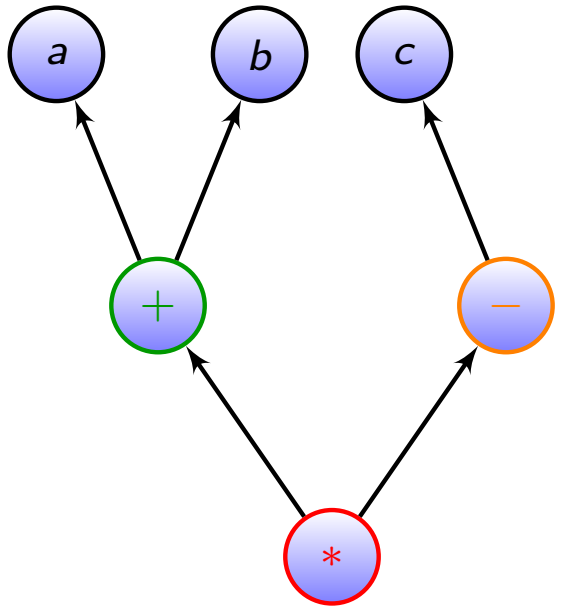
Drawbacks:

- Several variations on the same logic
- Special handling of the constant case

Goal:

- Minimize user required implementations
- Remove hand optimizations
- Add constant values to the expressions

```
w = (a + b) * (c - 4.0);
```



CoDiPack expression implementation - CoDiPack 2.0 approach

CoDiPack 2.0: 4 classes, 5 functions

- Classes: `UnaryExpression`, `BinaryExpression`, `LhsExpressionInterface`, `ConstantExpression`
- Functions: `getJacobian<link>`, `forEachLink`, `gradientA`, `gradientB`, `gradient`

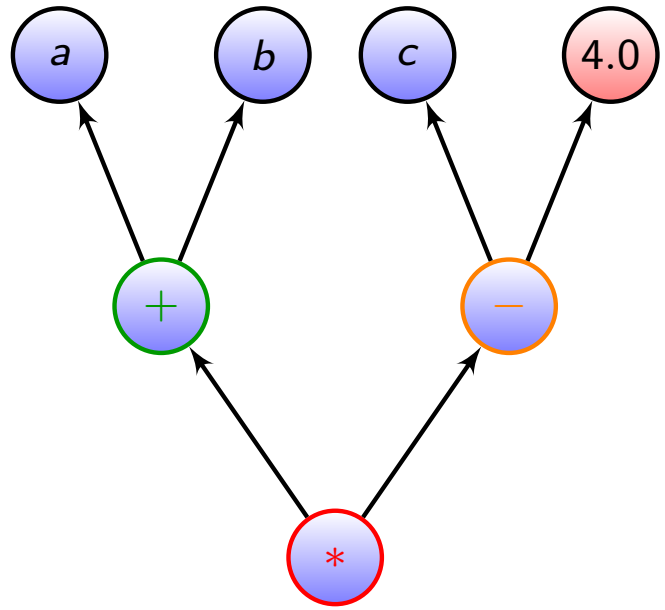
Advantages:

- New binary operator: Only two function implementations from user
- New expression: Only two function implementations from developer

Observations:

- No performance changes

```
w = (a + b) * (c - 4.0);
```



CoDiPack - Misc changes

- New folder Structure
  - Old: Grown over time, lots of files in single folder
  - New: More context specific folders, extra traits folder
- Declaration of `const` values:
  - `const` is a left binding operator
  - Right binding is legacy behavior
  - Wrong intuition for:

```
using T = int*;
using D1 = const T;  // Intuition: pointer to const int (Wrong)
using D2 = T const;  // const pointer to modifyable int
```

CoDiPack 2.0 programming guidelines - Template arguments

What about them?

- Hard to document - What range of types is allowed? (C++20 concepts)
- How to name them - With or without type suffix?
- How to declare them in the structure?
- No IDE support - auto completion
  - IDE does not know the type range
  - C++20 concept support will take quite a while.

CoDiPack 2.0 programming guidelines - Template arguments

What about them?

- Hard to document - What range of types is allowed? (C++20 concepts)
- How to name them - With or without type suffix?
- How to declare them in the structure?
- No IDE support - auto completion
  - IDE does not know the type range
  - C++20 concept support will take quite a while.

- CoDiPack 2.0 guideline:
  - Template arguments are declared with an underscore prefix
  - Template arguments are declared with `using` inside the structure without the prefix
  - They need to be declared with a default interface

CoDiPack 2.0 programming guidelines - Template arguments

Example:

```
template<typename _Tape>
struct ActiveType {
    using Tape = DECLARE_DEFAULT(
                        _Tape,
                        TEMPLATE(FullTapeInterface<double, double, ANY, ANY>));

    ...
};
```

- DECLARE_DEFAULT is a preprocessor switch
- In an IDE, the second argument is used
- Auto completion based on interface definition

CoDiPack 2.0 performance measurements - Burgers test case

Coupled Burgers equations test case:

$$u_t + uu_x + vu_y = \frac{1}{R}(u_{xx} + u_{yy})$$
$$v_t + uv_x + vv_y = \frac{1}{R}(v_{xx} + v_{yy})$$

- Evaluated on 601x601 grid with 32 time steps
- Run with two Intel Xeon SP 6126 CPUs on HPC cluster Elwetritsch of TU Kaiserslautern
- Two load cases:
  - Sequential (1 process on the node)
  - Bandwidth limited (24 process on the node)

CoDiPack 2.0 performance measurements - Offset computation

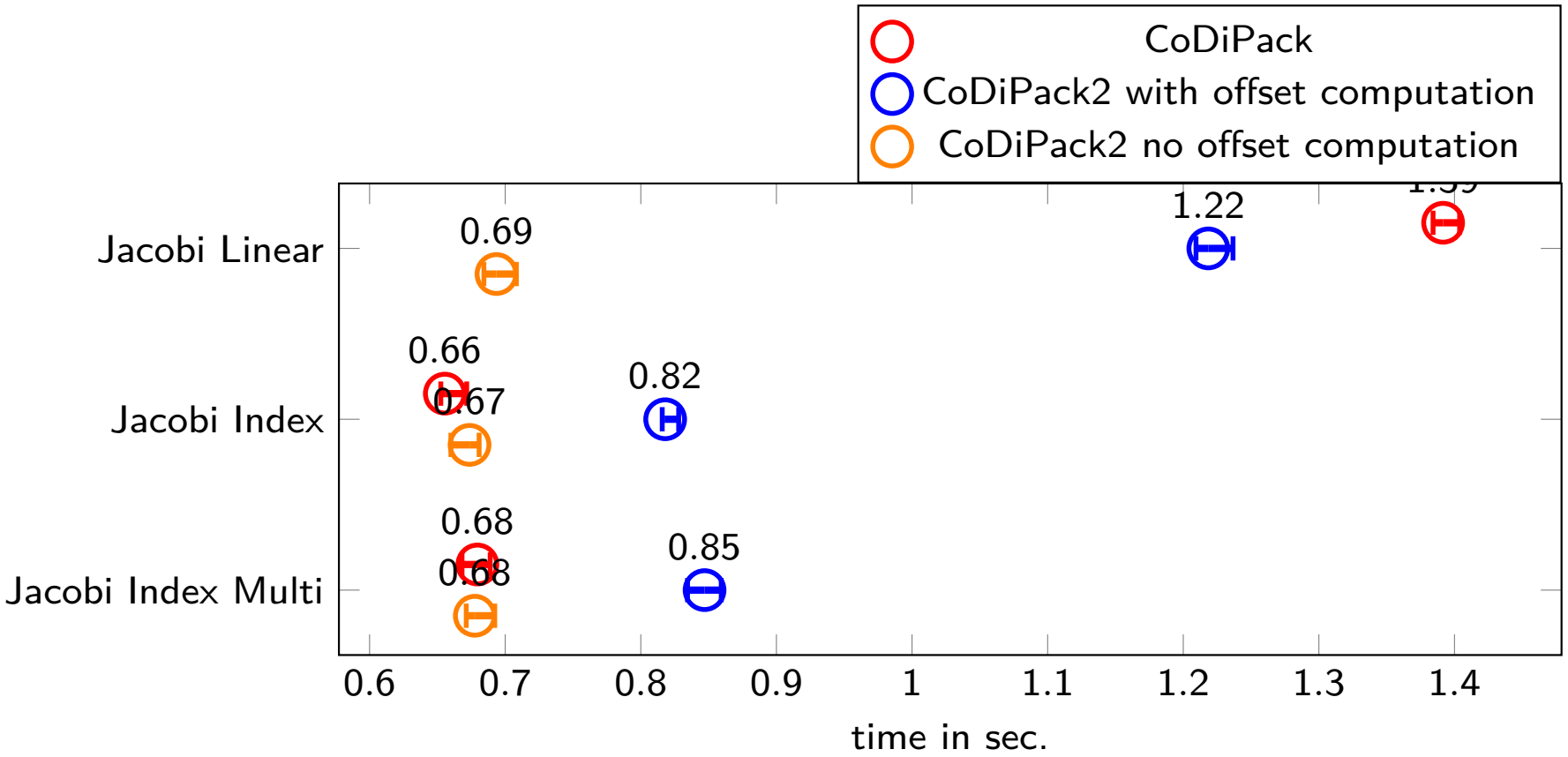
CoDiPack:

```
for(StatementInt curVar = 0; curVar < activeVariables; ++curVar) {
  --dataPos;
  adjoints[indices[dataPos]] += adj * jacobians[dataPos];
}
```

CoDiPack 2.0: (with offset computation)

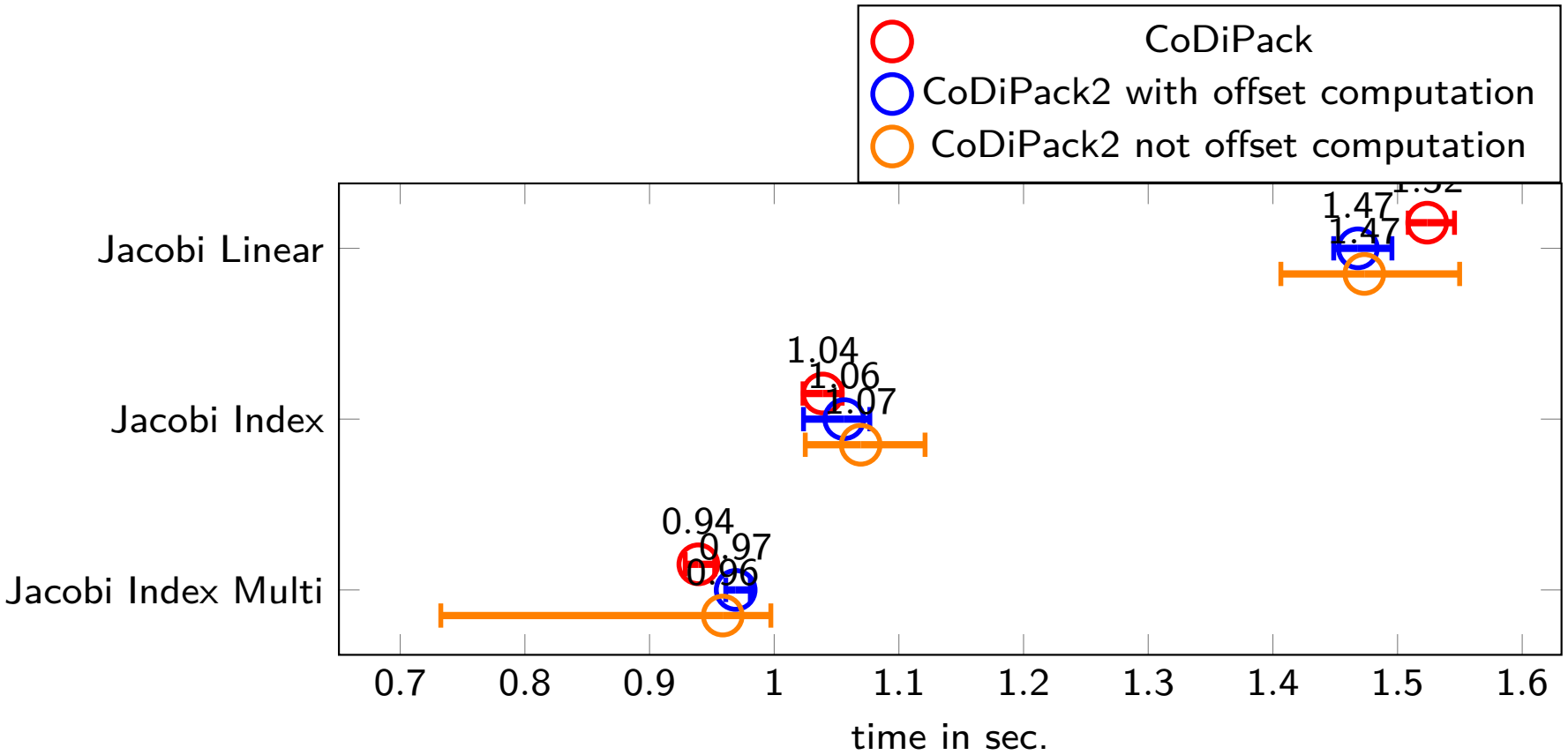
```
dataPos -= activeVariables;

for(StatementInt curVar = 0; curVar < activeVariables; curVar += 1) {
  size_t curOffset = dataPos + curVar;
  adjoints[indices[curOffset]] += adj * jacobians[curOffset];
}
```

CoDiPack 2.0 performance measurements - Offset computation



- Burgers test tape reversal sequential case

CoDiPack 2.0 performance measurements - Offset computation



- Burgers test tape reversal bandwidth limited case

CoDiPack 2.0 performance measurements - while loop optimization

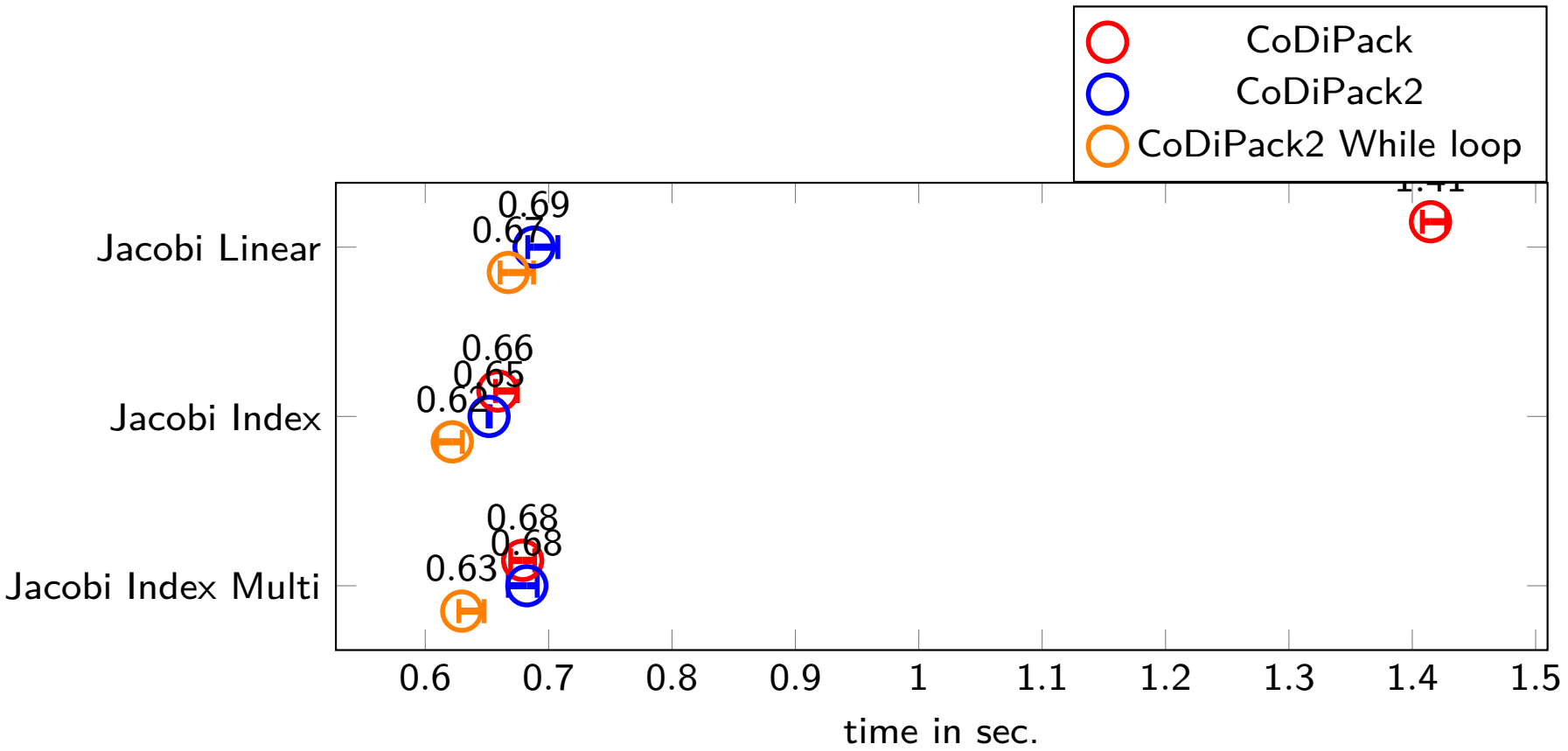
CoDiPack:

```
for(StatementInt curVar = 0; curVar < activeVariables; ++curVar) {
  --dataPos;
  adjoints[indices[dataPos]] += adj * jacobians[dataPos];
}
```

CoDiPack 2.0:

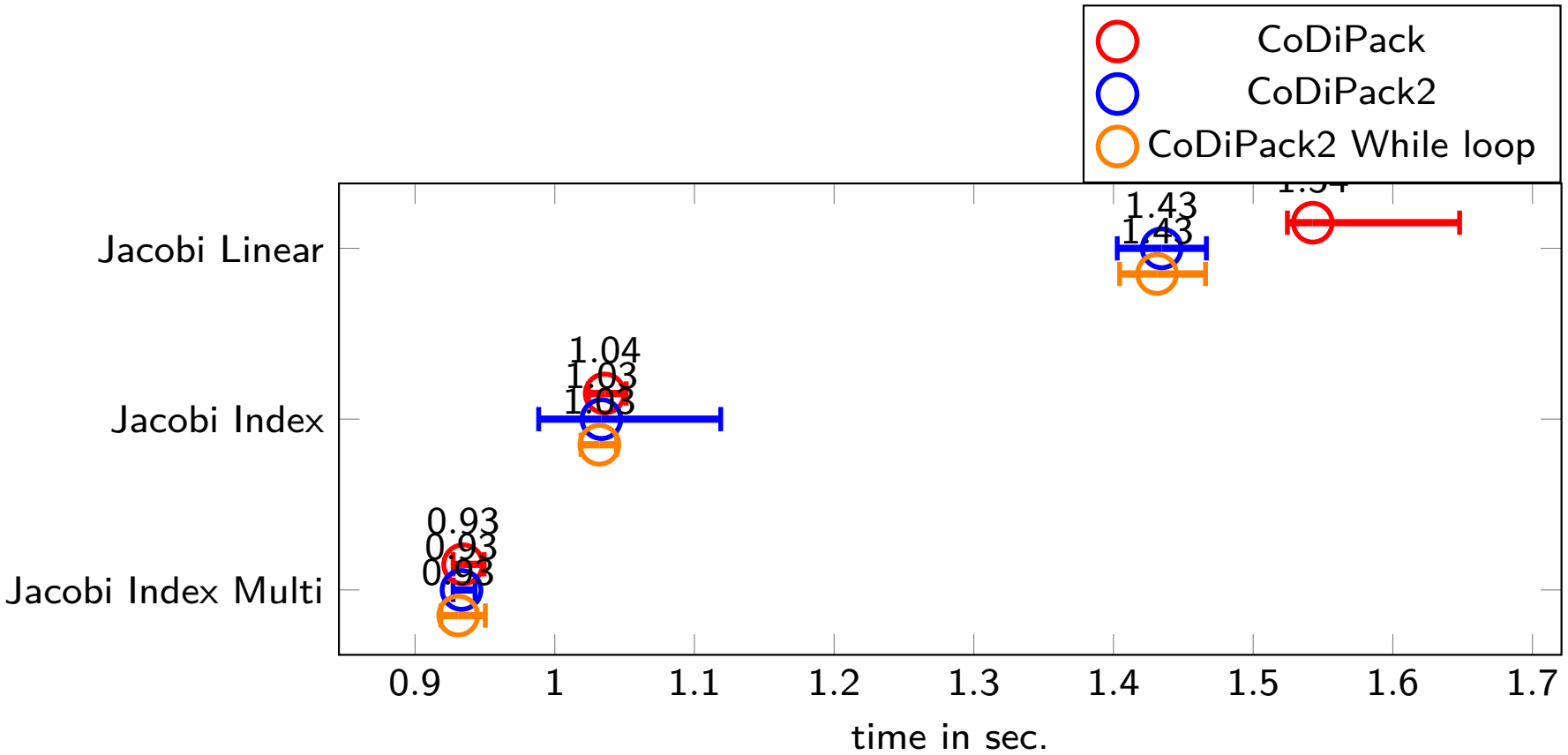
```
size_t endDataPos = dataPos - activeVariables;

while(endDataPos < dataPos) {
  dataPos -= 1;
  adjoints[indices[dataPos]] += adj * jacobians[dataPos];
}
```

CoDiPack 2.0 performance measurements - while loop optimization



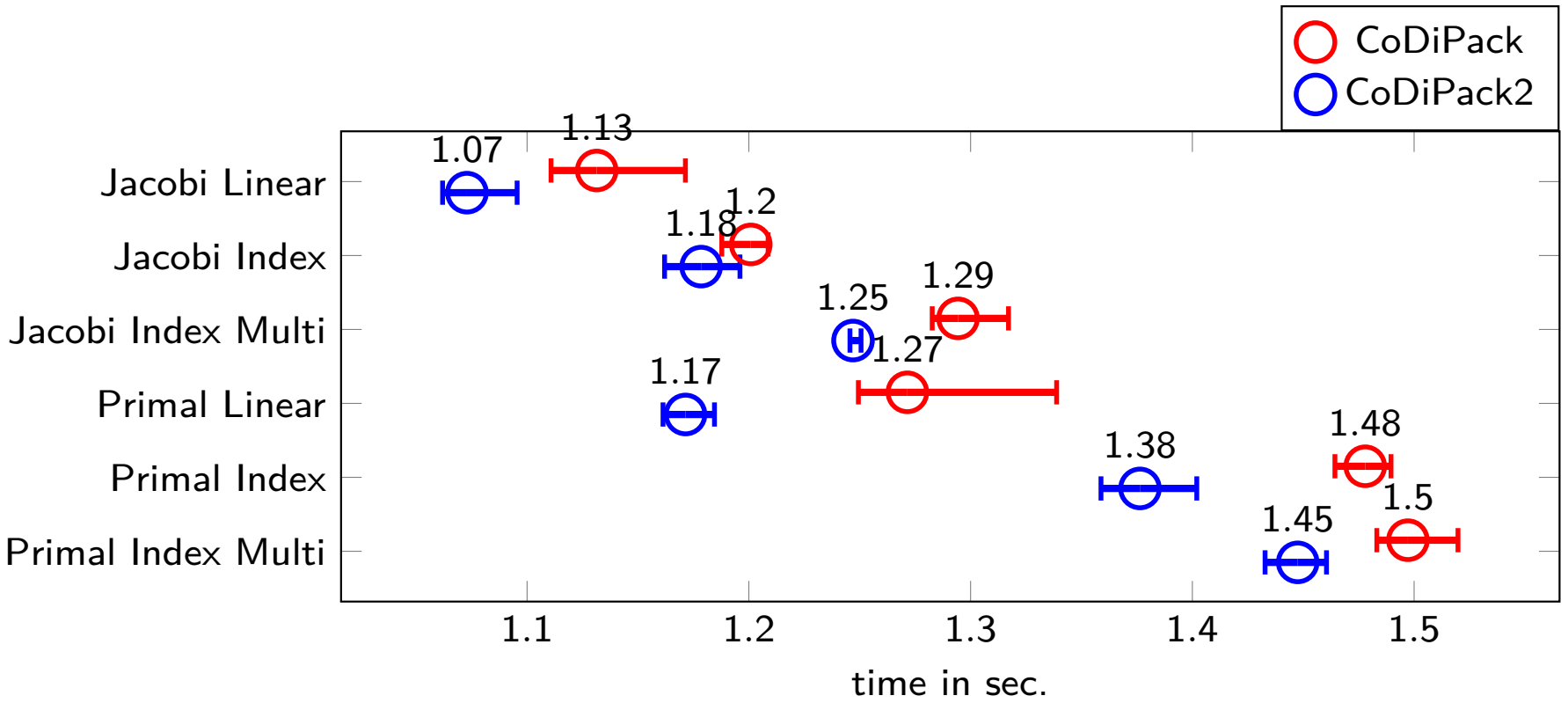
- Burgers test tape reversal sequential case

CoDiPack 2.0 performance measurements - while loop optimization



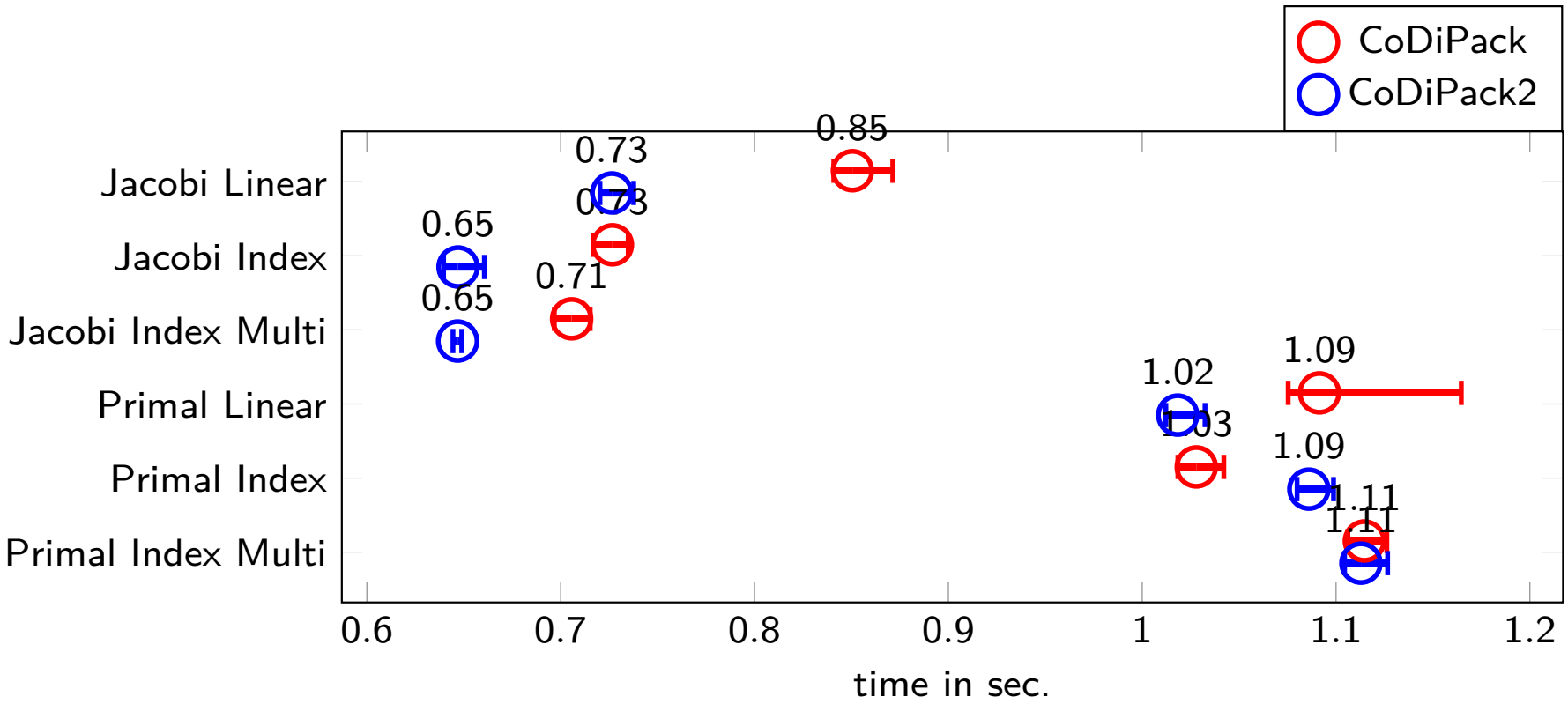
- Burgers test tape reversal bandwidth limited case

CoDiPack 2.0 performance measurements - current status



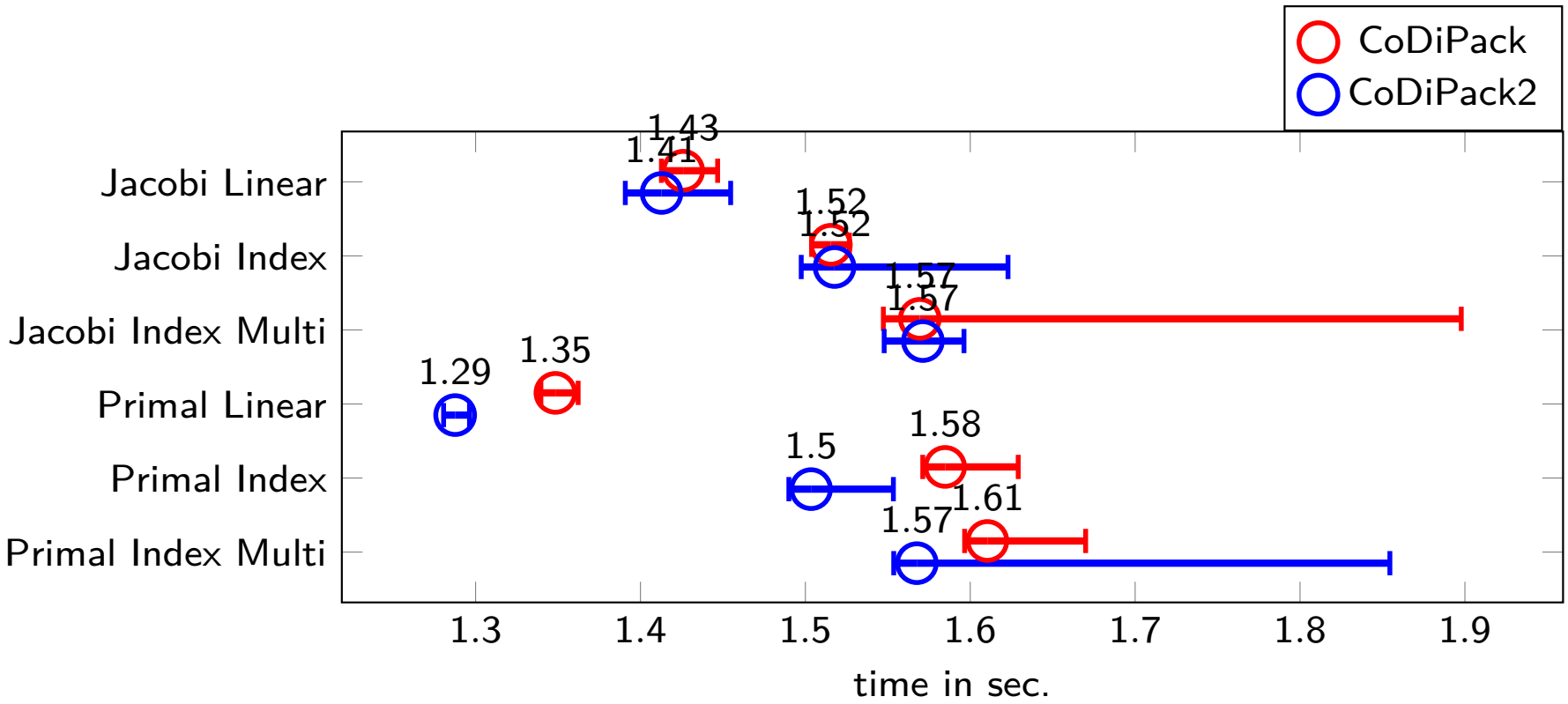
- Burgers test tape record sequential case

CoDiPack 2.0 performance measurements - current status



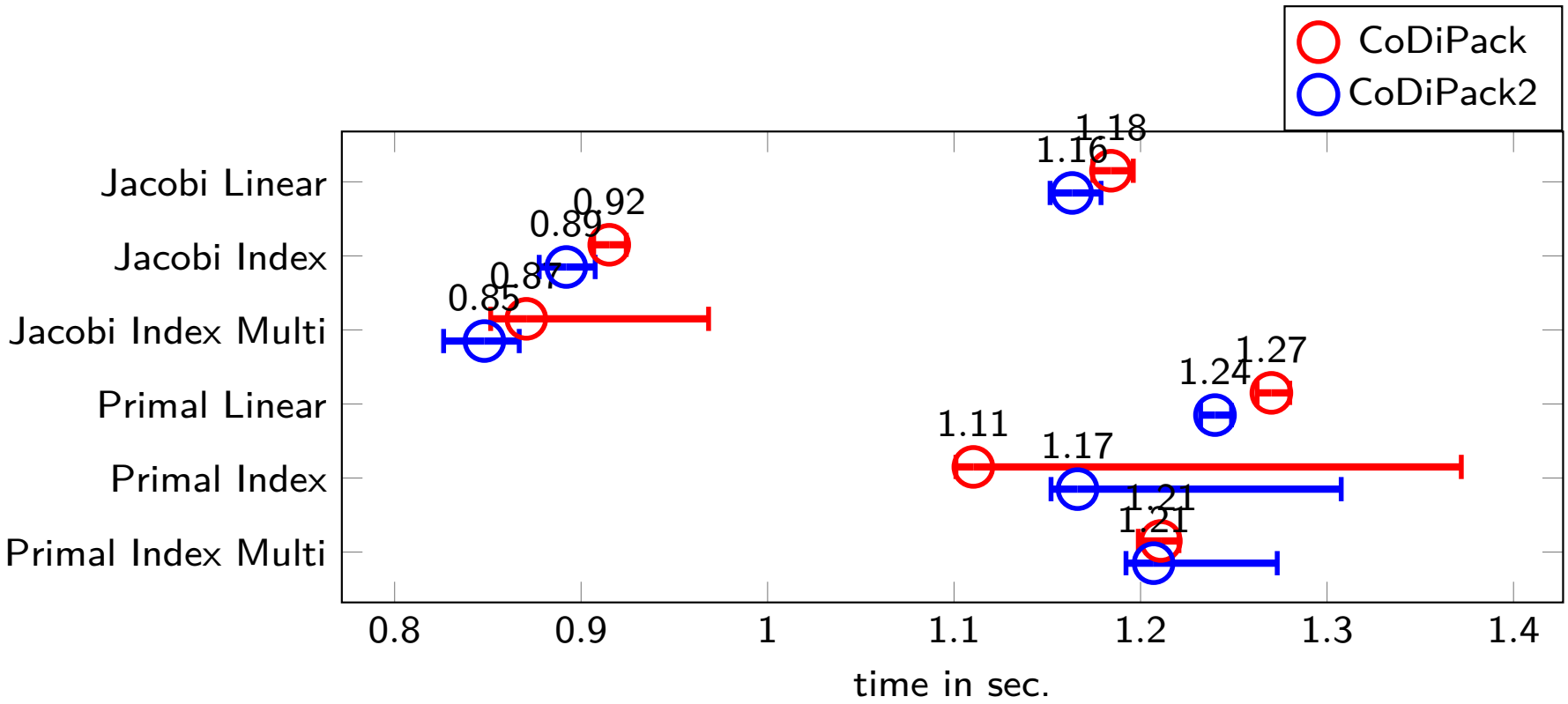
- Burgers test tape reversal sequential case

CoDiPack 2.0 performance measurements - current status



- Burgers test tape record bandwidth limited case

CoDiPack 2.0 performance measurements - current status



- Burgers test tape reversal bandwidth limited case

CoDiPack 2.0 - Conclusion & release

Conclusion:

- Performance gain in nearly all cases
- Simpler development of new tapes
- Simpler maintenance due to more code locality
- Consistent naming schemes
- Consistent coding style

Road map:

- Todo:
  - Port `EvaluationHelper`
  - User documentation
  - Developer documentation
- Release:
  - End of this year