Pseudo-Array Causationization in the New Backend

Karim Abdelhak, Bernhard Bachmann

University of Applied Sciences Bielefeld
Bielefeld, Germany

FH Bielefeld
University of
Applied Sciences
1. Overview
Overview

Back end Modules

Status on Array-Handling
Overview

Back end Modules

Status on Array-Handling

KArim Abdelhak, Bernhard Bachmann

Pseudo-Array Causalization

January 31, 2022
Overview

Backend Modules

Status on Array-Handling

Lowering → Simplify → Events → Partitioning → Causalize → Initialization → Categorize → Tearing → Jacobian → SimCode

- DetectStates
- Alias
- DAE-Mode

Finished | Partially Finished | Work in Progress
Overview

Backend Modules
Status on Array-Handling

Lowering → Simplify → Events → Partitioning → Causalize → Initialization → Tearing → Jacobian → SimCode

DetectStates → Alias → DAE-Mode

Initialization → Categorize

Finished → Partially Finished → Work in Progress
Backend Modules

Status on Array-Handling
2. Pseudo-Array Causalization
Algorithm Description

1. Create index mapping scalar $\leftrightarrow$ array for both variables and equations.
2. Create causalize modes for array equations, to recover array structure after causalization.
3. Create scalar Adjacency Matrix, do Matching and Sorting as if the system was scalarized.
4. Create buckets and fill them with the pseudo scalar equations. This collects all equations that belong to the same array equation and get causalized in the same way.
5. Create the BLT blocks with recovered array structures.
6. Slice the equations according to the bucket information (SimCode).
Algorithm Description

1. Create index mapping scalar $\leftrightarrow$ array for both variables and equations.
2. Create causalize modes for array equations, to recover array structure after causalization.
3. Create scalar Adjacency Matrix, do Matching and Sorting as if the system was scalarized.
4. Create buckets and fill them with the pseudo scalar equations. This collects all equations that belong to the same array equation and get causalized in the same way.
5. Create the BLT blocks with recovered array structures.
6. Slice the equations according to the bucket information (SimCode).
Algorithm Description

1. Create index mapping scalar $\leftrightarrow$ array for both variables and equations.
2. Create causalize modes for array equations, to recover array structure after causalization.
3. Create scalar Adjacency Matrix, do Matching and Sorting as if the system was scalarized.
4. Create buckets and fill them with the pseudo scalar equations. This collects all equations that belong to the same array equation and get causalized in the same way.
5. Create the BLT blocks with recovered array structures.
6. Slice the equations according to the bucket information (SimCode).
Algorithm Description

1. Create index mapping scalar $\leftrightarrow$ array for both variables and equations.
2. Create causalize modes for array equations, to recover array structure after causalization.
3. Create scalar Adjacency Matrix, do Matching and Sorting as if the system was scalarized.
4. Create buckets and fill them with the pseudo scalar equations. This collects all equations that belong to the same array equation and get causalized in the same way.
5. Create the BLT blocks with recovered array structures.
6. Slice the equations according to the bucket information (SimCode).
Algorithm Description

1. Create index mapping scalar <-> array for both variables and equations.
2. Create causalize modes for array equations, to recover array structure after causalization.
3. Create scalar Adjacency Matrix, do Matching and Sorting as if the system was scalarized.
4. Create buckets and fill them with the pseudo scalar equations. This collects all equations that belong to the same array equation and get causalized in the same way.
5. Create the BLT blocks with recovered array structures.
6. Slice the equations according to the bucket information (SimCode).
Algorithm Description

1. Create index mapping scalar $\leftrightarrow$ array for both variables and equations.
2. Create causalize modes for array equations, to recover array structure after causalization.
3. Create scalar Adjacency Matrix, do Matching and Sorting as if the system was scalarized.
4. Create buckets and fill them with the pseudo scalar equations. This collects all equations that belong to the same array equation and get causalized in the same way.
5. Create the BLT blocks with recovered array structures.
6. Slice the equations according to the bucket information (SimCode).
Algorithm Description

1. Create index mapping scalar $\leftrightarrow$ array for both variables and equations.
2. Create causalize modes for array equations, to recover array structure after causalization.
3. Create scalar Adjacency Matrix, do Matching and Sorting as if the system was scalarized.
4. Create buckets and fill them with the pseudo scalar equations. This collects all equations that belong to the same array equation and get causalized in the same way.
5. Create the BLT blocks with recovered array structures.
6. Slice the equations according to the bucket information (SimCode).
Structures in the New Backend
Index Mapping

```plaintext
record MAPPING
    "scalar <-> array index mapping for variables and equations"
    array<Integer> eqn_StA;
    array<Integer> var_StA;
    array<tuple<Integer,Integer>> eqn_AtS;
    array<tuple<Integer,Integer>> var_AtS;
end MAPPING;
```

Changes compared to the current Backend

1. Permanently store mapping information.
2. Also create a mapping for variables.
3. Save index lists as ranges (always consecutive).
4. Use universal utility functions to create scalar indices.
Index Mapping

```plaintext
record MAPPING
    "scalar <-> array index mapping for variables and equations"
    array< Integer > eqn_StA;
    array< Integer > var_StA;
    array< tuple< Integer, Integer > > eqn_AtS;
    array< tuple< Integer, Integer > > var_AtS;
end MAPPING;
```

Changes compared to the current Backend

1. Permanently store mapping information.
2. Also create a mapping for variables.
3. Save index lists as ranges (always consecutive).
4. Use universal utility functions to create scalar indices.
Structures in the New Backend

Index Mapping

```plaintext
record MAPPING
  "scalar <-> array index mapping for variables and equations"
  array<Integer> eqn_StA;
  array<Integer> var_StA;
  array<tuple<Integer, Integer>> eqn_AtS;
  array<tuple<Integer, Integer>> var_AtS;
end MAPPING;
```

Changes compared to the current Backend

1. Permanently store mapping information.
2. Also create a mapping for variables.
3. Save index lists as ranges (always consecutive).
4. Use universal utility functions to create scalar indices.
Structures in the New Backend

Index Mapping

```plaintext
record MAPPING
  "scalar <-> array index mapping for variables and equations"
  array< Integer > eqn_StA;
  array< Integer > var_StA;
  array< tuple< Integer, Integer >> eqn_AtS;
  array< tuple< Integer, Integer >> var_AtS;
end MAPPING;
```

Changes compared to the current Backend

1. Permanently store mapping information.
2. Also create a mapping for variables.
3. Save index lists as ranges (always consecutive).
4. Use universal utility functions to create scalar indices.
Structures in the New Backend

Index Mapping

```plaintext
record MAPPING
  "scalar <-> array index mapping for variables and equations"
  array<Integer> eqn_StA;
  array<Integer> var_StA;
  array<tuple<Integer,Integer>> eqn_AtS;
  array<tuple<Integer,Integer>> var_AtS;
end MAPPING;
```

Changes compared to the current Backend

1. Permanently store mapping information.
2. Also create a mapping for variables.
3. Save index lists as ranges (always consecutive).
4. Use universal utility functions to create scalar indices.
Pseudo-Array Causalization

Structures in the New Backend

Index Mapping

```plaintext
record MAPPING
  "scalar <-> array index mapping for variables and equations"
  array< Integer > eqn_StA;
  array< Integer > var_StA;
  array< tuple< Integer, Integer >> eqn_AtS;
  array< tuple< Integer, Integer >> var_AtS;
end MAPPING;
```

Changes compared to the current Backend

1. Permanently store mapping information.
2. Also create a mapping for variables.
3. Save index lists as ranges (always consecutive).
4. Use universal utility functions to create scalar indices.
Detecting the different ways an equation can be causalized.

1. Leave entries for scalar equations empty.
2. Create a mode index for each variable instance an array equation can be solved for.
3. Save the variable index to each mode index for each pseudo-scalarized equation (mode_to_var).
4. Save the variable instance to each mode index for each unscalarized equation (mode_to_cref).
Structures in the New Backend
Preparation the Slicing of Equations

record CAUSALIZE_MODES
  array<array<Integer>> mode_to_var;
  array<array<ComponentRef>> mode_to_cref;
end CAUSALIZE_MODES;

Detecting the different ways an equation can be causalized.

1. Leave entries for scalar equations empty.
2. Create a mode index for each variable instance an array equation can be solved for.
3. Save the variable index to each mode index for each pseudo-scalarized equation (mode_to_var).
4. Save the variable instance to each mode index for each unscalarized equation (mode_to_cref).
Structures in the New Backend
Preparing the Slicing of Equations

```plaintext
record CAUSALIZE_MODES
  "array equation reconstruction information"
  array<array<Integer>> mode_to_var;
  array<array<ComponentRef>> mode_to_cref;
end CAUSALIZE_MODES;
```

Detecting the different ways an equation can be causalized.

1. Leave entries for scalar equations empty.
2. Create a mode index for each variable instance an array equation can be solved for.
3. Save the variable index to each mode index for each pseudo-scalarized equation (mode_to_var).
4. Save the variable instance to each mode index for each unscalarized equation (mode_to_cref).
Detecting the different ways an equation can be causalized.

1. Leave entries for scalar equations empty.

2. Create a mode index for each variable instance an array equation can be solved for.

3. Save the variable index to each mode index for each pseudo-scalarized equation (mode_to_var).

4. Save the variable instance to each mode index for each unscalarized equation (mode_to_cref).
Structures in the New Backend
Preparing the Slicing of Equations

```plaintext
record CAUSALIZE_MODES
  "array equation reconstruction information"
  array<array<Integer>> mode_to_var;
  array<array<ComponentRef>> mode_to_cref;
end CAUSALIZE_MODES;
```

Detecting the different ways an equation can be causalized.

1. Leave entries for scalar equations empty.
2. Create a mode index for each variable `instance` an array equation can be solved for.
3. Save the variable index to each mode index for each pseudo-scalarized equation (`mode_to_var`).
4. Save the variable `instance` to each mode index for each unscalarized equation (`mode_to_cref`).
Structures in the New Backend
Preparing the Slicing of Equations

```plaintext
record CAUSALIZE_MODES
    "array equation reconstruction information"
    array<array<Integer>> mode_to_var;
    array<array<ComponentRef>> mode_to_cref;
end CAUSALIZE_MODES;
```

Detecting the different ways an equation can be causalized.

1. Leave entries for scalar equations empty.
2. Create a mode index for each variable instance an array equation can be solved for.
3. Save the variable index to each mode index for each pseudo-scalarized equation (mode_to_var).
4. Save the variable instance to each mode index for each unscalarized equation (mode_to_cref).
Recovering Array Structures

Bucket

```plaintext
record PSEUDO_BUCKET_KEY
    Integer eqn_arr_idx;
    Integer mode;
end PSEUDO_BUCKET_KEY;

record PSEUDO_BUCKET_VALUE
    ComponentRef cref_to_solve;
    list<Integer> eqn_scal_indices;
end PSEUDO_BUCKET_VALUE;

record PSEUDO_BUCKET
    "stores information about array equation subsets that get solved in the same way"
    UnorderedMap<PseudoBucketKey, PseudoBucketValue> bucket;
    array<Boolean> marks;
end PSEUDO_BUCKET;
```

Karem Abdelhak, Bernhard Bachmann

Pseudo-Array Causalization

January 31, 2022
Collecting Buckets

1. Traverse all strong components after scalar matching and sorting.
2. For each nontrivial strong component of size \( > 1 \) traverse all scalar equation indices.
3. Use the matching to get the matched scalar variable \( v \) for each scalar equation \( s \).
4. Use the \textit{mapping} and scalar equation \( s \) to get array equation \( a \).
5. Use the \textit{mode} \_to\_\textit{var} and matched variable \( v \) to get the causalization mode \( m \).
6. Use the \textit{mode} \_to\_\textit{cref} and mode \( m \) to get the cref \( c \).
7. If the \textit{key} = \( (a, m) \) exists, add the scalar index \( s \) to the list of \textit{value}, otherwise create \( \textit{value} = (c, \{s\}) \) and save it for that key.
Collecting Buckets

1. Traverse all strong components after scalar matching and sorting.
2. For each nontrivial strong component of size > 1 traverse all scalar equation indices.
3. Use the matching to get the matched scalar variable $v$ for each scalar equation $s$.
4. Use the $mapping$ and scalar equation $s$ to get array equation $a$.
5. Use the $mode_to_var$ and matched variable $v$ to get the causalization mode $m$.
6. Use the $mode_to_cref$ and mode $m$ to get the cref $c$.
7. If the key = $(a, m)$ exists, add the scalar index $s$ to the list of value, otherwise create $value = (c, \{s\})$ and save it for that key.
Collecting Buckets

1. Traverse all strong components after scalar matching and sorting.
2. For each nontrivial strong component of size > 1 traverse all scalar equation indices.
3. Use the matching to get the matched scalar variable $v$ for each scalar equation $s$.
4. Use the mapping and scalar equation $s$ to get array equation $a$.
5. Use the $mode\_to\_var$ and matched variable $v$ to get the causalization mode $m$.
6. Use the $mode\_to\_cref$ and mode $m$ to get the cref $c$.
7. If the $key = (a, m)$ exists, add the scalar index $s$ to the list of $value$, otherwise create $value = (c, \{s\})$ and save it for that key.
Recovering Array Structures

Collecting Buckets

1. Traverse all strong components after scalar matching and sorting.
2. For each nontrivial strong component of size > 1 traverse all scalar equation indices.
3. Use the matching to get the matched scalar variable \( v \) for each scalar equation \( s \).
4. Use the \( mapping \) and scalar equation \( s \) to get array equation \( a \).
5. Use the \( mode\_to\_var \) and matched variable \( v \) to get the causalization mode \( m \).
6. Use the \( mode\_to\_cref \) and mode \( m \) to get the cref \( c \).
7. If the \( key = (a, m) \) exists, add the scalar index \( s \) to the list of \( value \), otherwise create \( value = (c, \{s\}) \) and save it for that key.
Collecting Buckets

1. Traverse all strong components after scalar matching and sorting.
2. For each nontrivial strong component of size > 1 traverse all scalar equation indices.
3. Use the matching to get the matched scalar variable \( v \) for each scalar equation \( s \).
4. Use the \textit{mapping} and scalar equation \( s \) to get array equation \( a \).
5. Use the \textit{mode} \_\textit{to} \_\textit{var} and matched variable \( v \) to get the causalization mode \( m \).
6. Use the \textit{mode} \_\textit{to} \_\textit{cref} and mode \( m \) to get the cref \( c \).
7. If the \textit{key} = \((a, m)\) exists, add the scalar index \( s \) to the list of \textit{value}, otherwise create \textit{value} = \((c, \{s\})\) and save it for that key.
Collecting Buckets

1. Traverse all strong components after scalar matching and sorting.
2. For each nontrivial strong component of size > 1 traverse all scalar equation indices.
3. Use the matching to get the matched scalar variable $v$ for each scalar equation $s$.
4. Use the $mapping$ and scalar equation $s$ to get array equation $a$.
5. Use the $mode_to_var$ and matched variable $v$ to get the causalization mode $m$.
6. Use the $mode_to_cref$ and mode $m$ to get the cref $c$.
7. If the $key = (a, m)$ exists, add the scalar index $s$ to the list of $value$, otherwise create $value = (c, \{s\})$ and save it for that key.
Recovering Array Structures

Bucket

Collecting Buckets

1. Traverse all strong components after scalar matching and sorting.
2. For each nontrivial strong component of size > 1 traverse all scalar equation indices.
3. Use the matching to get the matched scalar variable $v$ for each scalar equation $s$.
4. Use the *mapping* and scalar equation $s$ to get array equation $a$.
5. Use the *mode_to_var* and matched variable $v$ to get the causalization mode $m$.
6. Use the *mode_to_cref* and mode $m$ to get the cref $c$.
7. If the key = $(a, m)$ exists, add the scalar index $s$ to the list of value, otherwise create
   $value = (c, \{s\})$ and save it for that key.
Collecting Buckets

1. Traverse all strong components after scalar matching and sorting.
2. For each nontrivial strong component of size > 1 traverse all scalar equation indices.
3. Use the matching to get the matched scalar variable \( v \) for each scalar equation \( s \).
4. Use the \textit{mapping} and scalar equation \( s \) to get array equation \( a \).
5. Use the \textit{mode\_to\_var} and matched variable \( v \) to get the causalization mode \( m \).
6. Use the \textit{mode\_to\_cref} and mode \( m \) to get the cref \( c \).
7. If the \( key = (a, m) \) exists, add the scalar index \( s \) to the list of \textit{value}, otherwise create \( value = (c, \{s\}) \) and save it for that key.
Recovering Array Structures

Slicing

Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable instance (cref).

BLOCK 3: Sliced Equation

### Variable:

\[ x[i, j] \]

### Equation:

\[
\text{FOR} \quad (8) (\$RES\_SIM\_2) \\
\text{for} \quad \{i \text{ in } 1:4, j \text{ in } 1:2\} \quad \text{loop} \\
[\text{SCAL} \quad (1) \quad x[i + 1, j] = x[i, j] - y[j]]; \\
\text{end for};
\]

with slices: \{2, 0, 3, 1\}
Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable instance (cref).

### BLOCK 3: Sliced Equation

#### Variable:

\[ x[i, j] \]

#### Equation:

\[
\begin{align*}
\text{FOR} \{i \text{ in } 1:4, \ j \text{ in } 1:2 \} & \text{ loop} \\
\text{SCAL} \{1\} \ x[i + 1, j] & = x[i, j] - y[j]; \\
\text{end for}; \\
\text{with slices: } \{2, 0, 3, 1\}
\end{align*}
\]
Recovering Array Structures

Slicing

Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable instance (cref).

BLOCK 3: Sliced Equation

### Variable:

\[ x[i, j] \]

### Equation:

\[
\begin{align*}
\text{FOR} & (8)(\text{RES\_SIM\_2}) \\
\text{for} & \{j \text{ in } 1:2, \ i \text{ in } 1:4\} \text{ loop} \\
\text{SCAL} & (1) x[i + 1, j] = x[i, j] - y[j]; \\
\text{end for} & \\
\text{with slices: } & \{2, 0, 3, 1\}
\end{align*}
\]
Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable instance (cref).

BLOCK 3: Sliced Equation

```plaintext
### Variable:
x[i, j]

### Equation:
[FOR-] (8)($RES_SIM_2)
[-----] for {j in 1:2, i in 1:4} loop
[-----] [SCAL] (1) x[i + 1, j] = x[i, j] - y[j];
[-----] end for;
with slices: {2, 0, 3, 1}
```
Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable instance (cref).

BLOCK 3: Sliced Equation

### Variable:

\[ x[i, j] \]

### Equation:

\[
\begin{align*}
&\text{[FOR−]} \ (8) \ ($RES\_SIM\_2$) \\
&\text{[-----] for } \{ j \text{ in } 1:2, \ i \text{ in } 1:2 \} \ \text{loop} \\
&\text{[-----] [SCAL] } (1) \ x[i + 1, j] = x[i, j] - y[j]; \\
&\text{[-----] end for;}
\end{align*}
\]

with slices: \{2, 0, 3, 1\}
Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable instance (cref).

### Variable:

\[ x[i, j] \]

### Equation:

\[
\text{[FOR]} \quad 8 \quad ($RES\_SIM\_2$) \\
\text{[-----] } \text{for } \{ j \text{ in } 1:2, \; i \text{ in } 1:2 \} \text{ loop} \\
\text{[-----] } \text{[SCAL]} \quad 1 \quad x[i + 1, \; j] = x[i, \; j] - y[j]; \\
\text{[-----] } \text{end for}; \\
\text{with slices: } \{2, \; 0, \; 3, \; 1\} \]
Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable instance (cref).

BLOCK 3: Sliced Equation

### Variable:
\[ x[i, j] \]

### Equation:
\[
\text{[FOR—]} \ (8) (\text{$RES\_SIM\_2$}) \\
\text{[———]} \ for \ \{ j \ in \ 1:2, \ i \ in \ 2:(-1):1 \} \ loop \\
\text{[———]} \ [SCAL] \ (1) \ x[i + 1, j] = x[i, j] - y[j]; \\
\text{[———]} \ end \ for; \\
\text{with \ slices:} \ \{2, 0, 3, 1\} \]
Recovering Array Structures

Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable instance (cref).

BLOCK 3: Sliced Equation

### Variable:

\[ x[i, j] \]

### Equation:

\[
\text{FOR} \quad (8) \quad ($RES\_SIM\_2$)
\]

\[
\text{for} \quad \{ j \text{ in } 1:2, \; i \text{ in } 2:(-1):1 \} \quad \text{loop}
\]

\[
\text{SCAL} \quad (1) \quad x[i + 1, j] = x[i, j] - y[j];
\]

\[
\text{end for;}
\]

with slices: \{2, 0, 3, 1\}
Recovering Array Structures

Slicing

Slicing Operations

1. Order the nested loop iterators properly.
2. Reduce the start and stop of the ranges according to the sliced subsets.
3. Fix the step of the ranges according to the sliced subsets.
4. Solve the body for the specified variable \texttt{instance} (cref).

\begin{verbatim}
BLOCK 3: Sliced Equation

### Variable:
\texttt{x[i, j]}

### Equation:
\texttt{\[ \begin{array}{c}
\text{FOR} \quad \{ j \text{ in } 1:2, \quad i \text{ in } 2:(-1):1 \} \quad \text{loop} \\
\text{SCAL} \quad (1) \quad \texttt{x[i, j]} = \texttt{x[i + 1, j]} + \texttt{y[j];}
\end{array} \]}

with slices: \{2, 0, 3, 1\}
\end{verbatim}
3. Exemplary Model
Exemplary Model

Modelica Model

```
model exemplary
  Real x[5,2];
  Real y[2];

equation
  for i in 1:4, j in 1:2 loop
    x[i+1,j] = x[i,j] - y[j];
  end for;
  for j in 1:2 loop
    y[j] = j*sin(time);
    x[3,j] = j*cos(time);
  end for;
end exemplary;
```

Expected Results

1. $y$ and $x[3,:]$ can be solved first, ordering does not matter.
2. $x[i+1,j]$ for $i > 3$ has to be solved after 1) in ascending order for $i$.
3. $x[i,j]$ for $i < 3$ has to be solved after 1) in descending order for $i$.
model exemplary
  Real x[5,2];
  Real y[2];
equation
  for i in 1:4, j in 1:2 loop
    x[i+1,j] = x[i,j] - y[j];
  end for;
  for j in 1:2 loop
    y[j] = j*sin(time);
    x[3,j] = j*cos(time);
  end for;
end exemplary;

Expected Results
1. y and x[3,:] can be solved first, ordering does not matter.
2. x[i+1,j] for i >= 3 has to be solved after 1) in ascending order for i.
3. x[i,j] for i < 3 has to be solved after 1) in descending order for i.
model exemplary
    Real x[5, 2];
    Real y[2];
equation
    for i in 1:4, j in 1:2 loop
        x[i+1,j] = x[i,j] - y[j];
    end for;
    for j in 1:2 loop
        y[j] = j*sin(time);
        x[3,j] = j*cos(time);
    end for;
end exemplary;

Expected Results

1. y and x[3,:] can be solved first, ordering does not matter.
2. x[i+1,j] for i >= 3 has to be solved after 1) in ascending order for i.
3. x[i,j] for i < 3 has to be solved after 1) in descending order for i.
Exemplary Model

Modelica Model

```model exemplary
    Real x[5,2];
    Real y[2];

equation
    for i in 1:4, j in 1:2 loop
        x[i+1,j] = x[i,j] - y[j];
    end for;
    for j in 1:2 loop
        y[j] = j*sin(time);
        x[3,j] = j*cos(time);
    end for;
end exemplary;
```

Expected Results

1. $y$ and $x[3,:]$ can be solved first, ordering does not matter.
2. $x[i+1,j]$ for $i \geq 3$ has to be solved after 1) in ascending order for $i$.
3. $x[i,j]$ for $i < 3$ has to be solved after 1) in descending order for $i$.
Exemplary Model

Modelica Model

```model exemplary
  Real x[5,2];
  Real y[2];
equation
  for i in 1:4, j in 1:2 loop
    x[i+1,j] = x[i,j] - y[j];
  end for;
  for j in 1:2 loop
    y[j] = j*sin(time);
    x[3,j] = j*cos(time);
  end for;
end exemplary;
```

Expected Results

1. $y$ and $x[3,:]$ can be solved first, ordering does not matter.
2. $x[i+1,j]$ for $i \geq 3$ has to be solved after 1) in ascending order for $i$.
3. $x[i,j]$ for $i < 3$ has to be solved after 1) in descending order for $i$.
Exemplary Model

Modelica Model

```model exemplary
  Real x[5, 2];
  Real y[2];
  equation
    for i in 1:4, j in 1:2 loop
      x[i+1,j] = x[i,j] - y[j];
    end for;
    for j in 1:2 loop
      y[j] = j*sin(time); 
      x[3,j] = j*cos(time);
    end for;
  end exemplary;

Expected Results

1. $y$ and $x_{[3,:]}$ can be solved first, ordering does not matter.
2. $x_{[i+1,j]}$ for $i \geq 3$ has to be solved after 1) in ascending order for $i$.
3. $x_{[i,j]}$ for $i < 3$ has to be solved after 1) in descending order for $i$.
### Exemplary Model

#### Block Lower Triangular Transformation (Unsolved)

---

**BLOCK 1: Sliced Equation**

```plaintext
### Variable:

| y[j] |

### Equation:

```plaintext
(FOR-) (2)($RES\_SIM\_1)

(----) for j in 1:2 loop

(----) [SCAL] (1) y[j] = CAST(Real, j) * sin(time);

(----) end for;

with slices: {0, 1}
```
Exemplary Model

Block Lower Triangular Transformation (Unsolved)

--------------

### Variable:

\[ x[3, j] \]

### Equation:

\[
\text{FOR}\quad (2) (\$RES\_SIM\_0)
\]

\[
\text{for } j \text{ in } 1:2 \text{ loop}
\]

\[
[\text{SCAL}] (1) x[3, j] = \text{CAST(Real, j)} \times \cos(\text{time});
\]

\[
\text{end for;}
\]

with slices: \{0, 1\}
Exemplary Model

Block Lower Triangular Transformation (Unsolved)

BLOCK 3: Sliced Equation

### Variable:

\[ x[i, j] \]

### Equation:

\[
\text{[FOR]} (8) (\$RES\_SIM\_2) \\
\text{[---]} \text{for} \{i \text{ in } 1:4, j \text{ in } 1:2\} \text{ loop} \\
\text{[---]} \text{[SCAL]} (1) x[i + 1, j] = x[i, j] - y[j]; \\
\text{[---]} \text{end for}; \\
\text{with} \text{ slices:} \{2, 0, 3, 1\} \]
**BLOCK 4: Sliced Equation**

```plaintext
### Variable:
\[ x[1 + i, j] \]

### Equation:
\[ \text{FOR} \{ i \text{ in } 1:4, j \text{ in } 1:2 \} \text{ loop} \]
\[ \text{SCAL} \{ 1 \} x[i + 1, j] = x[i, j] - y[j]; \]
\[ \text{end for} \]

with slices: \{ 4, 5, 6, 7 \}
```
Exemplary Model  
SimCode Equations (Solved)

for j in 1:2 loop
    y[j] := CAST(Real, j) * sin(time);
end for;

for j in 1:2 loop
    x[3, j] := CAST(Real, j) * cos(time);
end for;

for j in 1:1:2 loop
    for i in 2:(−1):1 loop
        x[i, j] := x[1 + i, j] + y[j];
    end for;
end for;

for i in 3:1:4 loop
    for j in 1:1:2 loop
        x[1 + i, j] := −(y[j] − x[i, j]);
    end for;
end for;
Exemplary Model

SimCode Equations (Solved)

```plaintext
for j in 1:2 loop
    y[j] := CAST(Real, j) * sin(time);
end for;

for j in 1:2 loop
    x[3, j] := CAST(Real, j) * cos(time);
end for;

for j in 1:1:2 loop
    for i in 2:(-1):1 loop
        x[i, j] := x[1 + i, j] + y[j];
    end for;
end for;

for i in 3:1:4 loop
    for j in 1:1:2 loop
        x[1 + i, j] := -(y[j] - x[i, j]);
    end for;
end for;
```
Exemplary Model
SimCode Equations (Solved)

```plaintext
for j in 1:2 loop
    y[j] := CAST(Real, j) * sin(time);
end for;

for j in 1:2 loop
    x[3, j] := CAST(Real, j) * cos(time);
end for;

for j in 1:1:2 loop
    for i in 2:(−1):1 loop
        x[i, j] := x[1 + i, j] + y[j];
    end for;
end for;

for i in 3:1:4 loop
    for j in 1:1:2 loop
        x[1 + i, j] := -(y[j] − x[i, j]);
    end for;
end for;
```
Exemplary Model
SimCode Equations (Solved)

\begin{align*}
\text{for } j \text{ in } 1:2 & \text{ loop} \\
& y[j] := \text{CAST(Real, j)} \ast \sin(\text{time}); \\
& \text{end for;}
\end{align*}

\begin{align*}
\text{for } j \text{ in } 1:2 & \text{ loop} \\
& x[3, j] := \text{CAST(Real, j)} \ast \cos(\text{time}); \\
& \text{end for;}
\end{align*}

\begin{align*}
\text{for } j \text{ in } 1:1:2 & \text{ loop} \\
& \text{for } i \text{ in } 2:(-1):1 \text{ loop} \\
& \quad x[i, j] := x[1 + i, j] + y[j]; \\
& \quad \text{end for;}
& \text{end for;}
\end{align*}

\begin{align*}
\text{for } i \text{ in } 3:1:4 & \text{ loop} \\
& \text{for } j \text{ in } 1:1:2 \text{ loop} \\
& \quad x[1 + i, j] := -(y[j] - x[i, j]); \\
& \quad \text{end for;}
& \text{end for;}
\end{align*}
4. Time Comparison
Scaled Exemplary Model
Modelica Model

model exemplaryS
  parameter Integer s = 0;
  Real x[5+s,2+s];
  Real y[2+s];
  equation
    for i in 1:4+s, j in 1:2+s loop
      x[i+1,j] = x[i,j] - y[j];
    end for;
    for j in 1:2+s loop
      y[j] = j*sin(time);
      x[3,j] = j*cos(time);
    end for;
end exemplaryS;
Scaled Exemplary Model

Comparison - Total Time

- New Backend
- Old Backend

Total Time (s)
scaling factor (squared)
Scaled Exemplary Model

Comparison - Backend Time

Time Comparison

Ka rim Abdelhak, Bernhard Bachmann
Scaled Exemplary Model

Comparison - Compilation Time

The graph compares the compilation time for the new backend and the old backend. The x-axis represents the scaling factor (squared), and the y-axis represents the compilation time (s). The new backend shows a linear increase in compilation time with the scaling factor, while the old backend shows a much steeper increase, indicating a more significant impact on compilation time as the scaling factor increases.
Scaled Exemplary Model
Comparison - Simulation Time

![Graph showing the comparison between New Backend and Old Backend in terms of Simulation Time for different scaling factors.](image-url)

- **New Backend**
- **Old Backend**
**ScalableTestSuite**

**Modelica Model**

```model CascadedFirstOrder
"N cascaded first order systems, approximating a pure delay"

parameter Integer N = 10 "Order of the system"

parameter Modelica.SIunits.Time T = 1 "System delay"

final parameter Modelica.SIunits.Time tau = T/N "Individual time constant"

Real x[N] (each start = 0, each fixed = true);

equation

  tau*der(x[1]) = 1 - x[1];

  for i in 2:N loop
    tau*der(x[i]) = x[i-1] - x[i];

end for;

end CascadedFirstOrder;
```
Scalable Test Suite
Comparison - Total Time

![Graph showing time comparison between New Backend and Old Backend](image-url)
Scalable Test Suite

Comparison - Backend Time

Scaling factor vs Backend Time (s)

- **New Backend**
- **Old Backend**

The graph shows a comparison of backend times for different scaling factors, with the New Backend consistently outperforming the Old Backend.
Scalable Test Suite
Comparison - Compilation Time
ScalableTestSuite
Comparison - Simulation Time

Simulation Time (s)

scaling factor

0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 2.2 2.4

0 1,000 2,000 3,000 4,000 5,000

New Backend
Old Backend

• Scaling factor

Simulation Time (s)

Scaling factor

0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2 2.2 2.4

0 1,000 2,000 3,000 4,000 5,000

New Backend
Old Backend

• Scaling factor
5. Summary
Summary

Recent Development

- Pseudo-Array matching and sorting for basic slices,
- Pseudo-Array handling for essential backend modules.

Current Development

- Pseudo-Array jacobians and sparsity pattern,
- Pseudo-Array handling for optimization modules,
- Mixed-kind variables,
- Better memory management.

Upcoming Plans

- Algorithm handling and function inlining,
- Pseudo-Array Index Reduction.
Summary

Recent Development
- Pseudo-Array matching and sorting for basic slices,
- Pseudo-Array handling for essential backend modules.

Current Development
- Pseudo-Array jacobians and sparsity pattern,
- Pseudo-Array handling for optimization modules,
- Mixed-kind variables,
- Better memory management.

Upcoming Plans
- Algorithm handling and function inlining,
- Pseudo-Array Index Reduction.
Recent Development
- Pseudo-Array matching and sorting for basic slices,
- Pseudo-Array handling for essential backend modules.

Current Development
- Pseudo-Array jacobians and sparsity pattern,
- Pseudo-Array handling for optimization modules,
- Mixed-kind variables,
- Better memory management.

Upcoming Plans
- Algorithm handling and function inlining,
- Pseudo-Array Index Reduction.
Thank you for your attention!