ILP-based Modulo Scheduling for High-level Synthesis

Julian Oppermann, Andreas Koch, Melanie Reuter-Oppermann, Oliver Sinnen
Outline

- Introduction to loop pipelining / modulo scheduling
Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches
Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches
Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches

Modulo SDC
Canis et al.

Formulation by
Eichenberger &
Davidson

state-of-the-art heuristic state-of-the-art exact formulation
Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches

Modulo SDC
Canis et al.

Formulation by
Eichenberger & Davidson

Moovac
Oppermann et al.

state-of-the-art heuristic  state-of-the-art exact formulation  novel exact formulation
Outline

- Introduction to loop pipelining / modulo scheduling
- Comparison of a novel & two existing approaches
  - result quality, heuristic vs. exact

- Modulo SDC
  Canis et al.

- State-of-the-art heuristic

- Formulation by Eichenberger & Davidson

- State-of-the-art exact formulation

- Moovac
  Oppermann et al.

- Novel exact formulation
Introduction to loop pipelining / modulo scheduling

Comparison of a novel & two existing approaches

- result quality, heuristic vs. exact
- **time to schedule** - *it’s impractical to do exact modulo scheduling, right?*

- **Modulo SDC**
  Canis et al.
  state-of-the-art **heuristic**

- **Formulation by Eichenberger & Davidson**
  state-of-the-art **exact** formulation

- **Moovac**
  Oppermann et al.
  novel **exact** formulation
C-based High-level Synthesis (HLS) needs to exploit all sources of parallelism
Loop Pipelining

- **C-based High-level Synthesis (HLS)** needs to exploit all sources of parallelism

- **Loop pipelining**
  = new loop iterations are started after a fixed number of time steps, called **Initiation Interval (II)**
  
  - Partially overlapping execution of subsequent loop iterations
- Increases throughput!
Loop Pipelining

- Increases throughput!

- Executing $n$ iterations →
  
  $n \cdot SL$  time steps w/o pipelining
  
  $(n-1) \cdot II + SL$  time steps with pipelining
Loop Pipelining

- Increases throughput!

- Executing $n$ iterations →
  \[ n \cdot SL \] time steps w/o pipelining
  \[ (n-1) \cdot II + SL \] time steps with pipelining

- Primary objective is to find **smallest feasible II**
  - Limited by dependencies between iterations
  - Subject to resource constraints (cache ports, DSPs, …)
- Operations from different iterations are active at the same time
Loop Pipelining

- Operations from different iterations are active at the same time
- Resource constraints have to hold for congruence classes (modulo II) of time steps

  - “modulo resource constraints”
Loop Pipelining

- Operations from different iterations are active at the same time
- Resource constraints have to hold for congruence classes (modulo $II$) of time steps
  - “modulo resource constraints”
- Suitable schedules for loop pipelining are found by modulo schedulers
Example

```c
for (i = 1 .. N) {
    t = a[i-1];
    a[i] = s + t;
    s = t * t;
}
```
for (i = 1 .. N) {
    t = a[i-1];
    a[i] = s + t;
    s = t * t;
}

data flow implies precedence constraints
Example

\begin{verbatim}
for (i = 1 .. N)
{
    t = a[i-1];
    a[i] = s + t;
    s = t * t;
}
\end{verbatim}

add operation depends on the value of s from the previous iteration
Example

```c
for (i = 1 .. N) {
    t = a[i-1];
    a[i] = s + t;
    s = t * t;
}
```

- Load value only after it was written in the previous iteration
- Add operation depends on the value of s from the previous iteration
Example

for (i = 1 .. N)
{
    t = a[i-1];
    a[i] = s + t;
    s = t * t;
}

add operation depends on the value of s from the previous iteration

Both edges imply inter-iteration dependencies a.k.a “backedges”

Load value only after it was written in the previous iteration
for (i = 1 .. N)
{
    t = a[i-1];
    a[i] = s + t;
    s = t * t;
}

<table>
<thead>
<tr>
<th>Time step</th>
<th>Iteration 0 / modulo schedule</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ld a[i-1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>st a[i]</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>st a[i]</td>
<td>ld a[i-1]</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
General Approach

- Determine lower and upper bound for the II
General Approach

- Determine lower and upper bound for the II
- Try to find a feasible modulo schedule
General Approach

- Determine lower and upper bound for the II
- Try to find a feasible modulo schedule
  - Input: candidate II, precedence edges, resource constraints, operation latencies
General Approach

- Determine lower and upper bound for the II
- Try to find a feasible modulo schedule
  - Input: candidate II, precedence edges, resource constraints, operation latencies
  - Output: start times for operations, or attempt fails
General Approach

- Here: Compare schedulers based on **Integer Linear Programs** (ILP)
General Approach

- Here: Compare schedulers based on Integer Linear Programs (ILP)

- Scheduling graphs with only typical HLS precedence constraints and backedges is easy
  
  - e.g. as a System of Difference Constraints (SDC), special ILP that can be solved in polynomial time
General Approach

- Here: Compare schedulers based on **Integer Linear Programs (ILP)**

- Scheduling graphs with only typical HLS precedence constraints and backedges is **easy**
  - e.g. as a System of Difference Constraints (SDC), special ILP that can be solved in **polynomial** time

- Approaches differ in the modelling of resource constraints
General Approach

- $A_k$ instances/units/... of a certain scarce resource kind $k$
General Approach

- $A_k$ instances/units/… of a certain scarce resource kind $k$
- Candidate II $\Rightarrow$ congruence classes of operations’ start times
### General Approach

- $A_k$ instances/units/… of a certain scarce resource kind $k$
- Candidate II $\Rightarrow$ congruence classes of operations’ start times
- Each instance can be used once per congruence class by an operation $i$

<table>
<thead>
<tr>
<th>Congruence classes</th>
<th>Resource instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$i$</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>II-1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$A_k$-1</td>
</tr>
</tbody>
</table>
General Approach

- $A_k$ instances/units/… of a certain scarce resource kind $k$
- Candidate II $\Rightarrow$ congruence classes of operations’ start times
- Each instance can be used once per congruence class by an operation $i$
- “modulo reservation table” (MRT)
- **Heuristic** using an SDC and an explicit MRT

\[
\begin{align*}
\text{min} & \ldots \\
\text{s.t.} & \quad v_j - v_i \leq 1 \\
& \quad \ldots 
\end{align*}
\]
- **Heuristic** using an SDC and an explicit MRT
  - Start with a resource-\textbf{un}constrained schedule

\[ \text{min} \ldots \\
\text{s.t. } v_j - v_i \leq 1 \\
\ldots \]
- **Heuristic** using an SDC and an explicit MRT
  
  - Start with a resource-**un**constrained schedule
  
  - Incrementally try to assign operations to MRT and add constraints to SDC

\[
\begin{align*}
\text{min} & \quad \cdots \\
\text{s.t.} & \quad v_j - v_i \leq 1 \\
& \quad \cdots
\end{align*}
\]
Modulo SDC

- **Heuristic** using an SDC and an explicit MRT
  - Start with a resource-unconstrained schedule
  - Incrementally try to assign operations to MRT and add constraints to SDC
  - Backtracking required if SDC becomes infeasible

\[
\begin{align*}
\min & \quad \ldots \\
\text{s.t.} & \quad v_j - v_i \leq 1 \\
& \quad \ldots
\end{align*}
\]
Heuristic using an SDC and an explicit MRT

- Start with a resource-unconstrained schedule
- Incrementally try to assign operations to MRT and add constraints to SDC
- Backtracking required if SDC becomes infeasible
- Successful if all resource-constrained ops fit in MRT

\[
\min \ldots \\
\text{s.t.} \\
\quad v_j - v_i \leq 1 \\
\quad \ldots
\]
**Eichenberger’s Formulation**

- **Exact** formulation
  general ILP with time-indexed binary variables
  \( a_{m,i} := “\text{operation } i \text{ starts in congruence class } m” \)

<table>
<thead>
<tr>
<th>Congruence classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_{0,0} )</td>
</tr>
<tr>
<td>( a_{1,0} )</td>
</tr>
<tr>
<td>( a_{2,0} )</td>
</tr>
<tr>
<td>( a_{\ll-1,0} )</td>
</tr>
<tr>
<td>( a_{0,i} )</td>
</tr>
<tr>
<td>( a_{1,i} )</td>
</tr>
<tr>
<td>( a_{2,i} )</td>
</tr>
<tr>
<td>( a_{\ll-1,i} )</td>
</tr>
<tr>
<td>( a_{0,j} )</td>
</tr>
<tr>
<td>( a_{1,j} )</td>
</tr>
<tr>
<td>( a_{2,j} )</td>
</tr>
<tr>
<td>( a_{\ll-1,j} )</td>
</tr>
<tr>
<td>( a_{0,N} )</td>
</tr>
<tr>
<td>( a_{1,N} )</td>
</tr>
<tr>
<td>( a_{2,N} )</td>
</tr>
<tr>
<td>( a_{\ll-1,N} )</td>
</tr>
</tbody>
</table>

Operations: 0, i, j, N
Eichenberger’s Formulation

- **Exact** formulation
  general ILP with time-indexed binary variables
  \( a_{m,i} := \text{“operation } i \text{ starts in congruence class } m \” \)

- Example: Resource constraint for kind \( k \), congruence class 2
  fulfilled iff.

  \[ \sum_x a_{2,x} \leq A_k \]

  for all operations \( x \) that use a \( k \)-resource
Moovac

- Moovac = Modulo Overlap Variable Constraints

- Adapted task scheduling formulation based on overlap variables
- Moovac = **Modulo Overlap Variable Constraints**
- Adapted task scheduling formulation based on overlap variables
- **Exact** formulation, general ILP
Moovac

- Moovac = **Modulo Overlap Variable Constraints**
- Adapted task scheduling formulation based on overlap variables
- **Exact** formulation, general ILP
- **Integer** variables model start times $t_i$
Let $i, j$ be operations that require a resource of kind $k$
Let $i, j$ be operations that require a resource of kind $k$

Resource assignment modelled by

- Integer variables
  - $r_i$ resource instance ID $\in [0 \ldots A_k - 1]$
  - $m_i$ congruence class ID $\in [0 \ldots \text{candidate II} - 1]$
Let $i$, $j$ be operations that require a resource of kind $k$

Resource assignment modelled by

- Integer variables
  - $r_i$ resource instance ID $\in [0 \ldots A_k - 1]$
  - $m_i$ congruence class ID $\in [0 \ldots \text{candidate ll} - 1]$

- Binary overlap variables
  - $\varepsilon_{ij} := 1$ iff. $r_i < r_j$
  - $\mu_{ij} := 1$ iff. $m_i < m_j$
Let $i, j$ be operations that require a resource of kind $k$

Resource assignment modelled by

- Integer variables
  - $r_i$ resource instance ID $\in [0 \ldots A_k - 1]$
  - $m_i$ congruence class ID $\in [0 \ldots \text{candidate II} - 1]$

- Binary overlap variables
  - $\varepsilon_{ij} := 1$ iff. $r_i < r_j$
  - $\mu_{ij} := 1$ iff. $m_i < m_j$

No resource conflict iff.

$$\varepsilon_{ij} + \varepsilon_{ji} + \mu_{ij} + \mu_{ji} \geq 1$$

“$i$ and $j$ are either assigned to different resource instances, or scheduled to different congruence classes”
- Tuples \((m_i, r_j) \Rightarrow \text{cell in MRT for operation } i\)
- Tuples \((m_i, r_j)\) \(\Rightarrow\) cell in MRT for operation \(i\)

- Overlap variables model relations between operations
- Tuples \((m_i, r_i)\) ⇒ cell in MRT for operation \(i\)

- Overlap variables model relations between operations

\[ \mu_{uv} = 1 \]
- Tuples \((m_i, r_j)\) \(\Rightarrow\) cell in MRT for operation \(i\)

- Overlap variables model relations between operations

\[
\begin{align*}
\mu_{uv} &= 1 \\
\varepsilon_{uw} &= 1 \\
\mu_{uw} &= 1
\end{align*}
\]
- Tuples $(m_i, r_j) \Rightarrow \text{cell in MRT for operation } i$

- Overlap variables model relations between operations

\[ \mu_{uv} = 1 \]

\[ \varepsilon_{uw} = 1 \]

\[ \mu_{uw} = 1 \]

\[ \varepsilon_{vw} = 1 \]
Approaches At A Glance

Modulo SDC
Canis et al.

- Resource constraints are not part of the linear program
- Operations are assigned *heuristically* to MRT
Approaches At A Glance

Modulo SDC
Canis et al.

- Resource constraints are not part of the linear program
- Operations are assigned **heuristically** to MRT

Formulation by Eichenberger & Davidson

- **Exact** formulation
- Time-indexing → large number of binary variables, complicated constraints
Approaches At A Glance

- **Modulo SDC**
  - Canis et al.
  - Resource constraints are not part of the linear program
  - Operations are assigned *heuristically* to MRT

- **Formulation by Eichenberger & Davidson**
  - Exact formulation
  - Time-indexing $\rightarrow$ large number of binary variables, complicated constraints

- **Moovac**
  - Oppermann et al.
  - Novel *exact* formulation
  - Uses fewer integer variables and overlap variables to model inequality between them
Evaluation

- Schedulers implemented with CPLEX 12.6.3
Evaluation

- Schedulers implemented with CPLEX 12.6.3
- Single-threaded execution on Intel Xeon E5-2667’s at 3.3 GHz
Evaluation

- Schedulers implemented with CPLEX 12.6.3
- Single-threaded execution on Intel Xeon E5-2667’s at 3.3 GHz
- Time limit of 5 min or 60 min per candidate II
  - increment II if no solution was found
Evaluation

- Schedulers implemented with CPLEX 12.6.3
- Single-threaded execution on Intel Xeon E5-2667’s at 3.3 GHz
- Time limit of 5 min or 60 min per candidate II
  - increment II if no solution was found
- Attempted to schedule 225 graphs from CHStone and MachSuite
  - up to 1124 operations / up to 107 resource-constrained operations
### Results (Quality)

- **5 min time limit**

<table>
<thead>
<tr>
<th>Graphs</th>
<th>Moovac vs. Modulo SDC</th>
<th>Moovac vs. Eichenberger’s ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shorter II found by …</td>
<td>shorter II found by …</td>
</tr>
<tr>
<td>Size</td>
<td>#</td>
<td>Moovac</td>
</tr>
<tr>
<td>all</td>
<td>225</td>
<td>6</td>
</tr>
<tr>
<td>small</td>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>large</td>
<td>22</td>
<td>5</td>
</tr>
</tbody>
</table>
5 min time limit

### Results (Quality)

<table>
<thead>
<tr>
<th></th>
<th>Moovac vs. Modulo SDC</th>
<th>Moovac vs. Eichenberger’s ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shorter II found by ...</td>
<td>shorter II found by ...</td>
</tr>
<tr>
<td>Size</td>
<td>Moovac</td>
<td>Same</td>
</tr>
<tr>
<td>all</td>
<td>225</td>
<td>6</td>
</tr>
<tr>
<td>small</td>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>large</td>
<td>22</td>
<td>5</td>
</tr>
</tbody>
</table>

Modulo SDC delivers high-quality results
Results (Quality)

- 5 min time limit

<table>
<thead>
<tr>
<th>Graphs</th>
<th>Moovac vs. Modulo SDC</th>
<th>Moovac vs. Eichenberger’s ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>shorter II found by ...</td>
<td>shorter II found by ...</td>
</tr>
<tr>
<td>Size</td>
<td>#</td>
<td>Moovac</td>
</tr>
<tr>
<td>all</td>
<td>225</td>
<td>6</td>
</tr>
<tr>
<td>small</td>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>large</td>
<td>22</td>
<td>5</td>
</tr>
</tbody>
</table>

Modulo SDC delivers high-quality results

Modulo SDC found schedules where Moovac ran out of time
## Results (Quality)

- **5 min time limit**

<table>
<thead>
<tr>
<th>Graphs</th>
<th>Moovac vs. Modulo SDC</th>
<th>Moovac vs. Eichenberger’s ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>shorter II found by …</strong></td>
<td><strong>shorter II found by …</strong></td>
</tr>
<tr>
<td>Size</td>
<td>#</td>
<td>Moovac</td>
</tr>
<tr>
<td>all</td>
<td>225</td>
<td>6</td>
</tr>
<tr>
<td>small</td>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>large</td>
<td>22</td>
<td>5</td>
</tr>
</tbody>
</table>

Modulo SDC delivers high-quality results

Exact schedulers should find same II, but E.B. hit time limit

Modulo SDC found schedules where Moovac ran out of time
## Results (Time)

- Scheduling duration with 5 min time limit:

<table>
<thead>
<tr>
<th>Graphs Size</th>
<th>#</th>
<th>Moovac</th>
<th>Modulo SDC</th>
<th>Eichenberger’s ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>225</td>
<td>489</td>
<td>96</td>
<td>753</td>
</tr>
<tr>
<td>small</td>
<td>203</td>
<td>3</td>
<td>0</td>
<td>131</td>
</tr>
<tr>
<td>large</td>
<td>22</td>
<td>486</td>
<td>96</td>
<td>623</td>
</tr>
</tbody>
</table>
Results (Time)

- Scheduling duration with 5 min time limit:

<table>
<thead>
<tr>
<th>Graphs Size</th>
<th>#</th>
<th>Moovac</th>
<th>Modulo SDC</th>
<th>Eichenberger’s ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>225</td>
<td>489</td>
<td>96</td>
<td>753</td>
</tr>
<tr>
<td>small</td>
<td>203</td>
<td>3</td>
<td>0</td>
<td>131</td>
</tr>
<tr>
<td>large</td>
<td>22</td>
<td>486</td>
<td>96</td>
<td>623</td>
</tr>
</tbody>
</table>

Moovac is faster than the other approaches
Results (Time)

- Scheduling duration with 5 min time limit:

<table>
<thead>
<tr>
<th>Graphs</th>
<th>Size</th>
<th>#</th>
<th>Moovac</th>
<th>Modulo SDC</th>
<th>Eichenberger’s ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>all</td>
<td>225</td>
<td>489</td>
<td>96</td>
<td>753</td>
</tr>
<tr>
<td>small</td>
<td>small</td>
<td>203</td>
<td>3</td>
<td>0</td>
<td>131</td>
</tr>
<tr>
<td>large</td>
<td>large</td>
<td>22</td>
<td>486</td>
<td>96</td>
<td>623</td>
</tr>
</tbody>
</table>

Moovac is faster than the other approaches

The timeouts dominate the overall time e.g. 96 x 5 min = 480 min
## Results (Time)

- **Scheduling duration with 5 min time limit:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>225</td>
<td>489</td>
<td>96</td>
<td>753</td>
<td>148</td>
<td>932</td>
<td>177</td>
</tr>
<tr>
<td>small</td>
<td>203</td>
<td>3</td>
<td>0</td>
<td>131</td>
<td>26</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>large</td>
<td>22</td>
<td>486</td>
<td>96</td>
<td>623</td>
<td>122</td>
<td>927</td>
<td>177</td>
</tr>
</tbody>
</table>

Moovac is faster than the other approaches.

The timeouts dominate the overall time e.g. 96 x 5 min = 480 min.

M. SDC seems to get stuck even on small graphs.
Insights

- How can an exact formulation be faster overall than the heuristic?
Insights

- How can an exact formulation be faster overall than the heuristic?
  - ILP solver “sees” whole problem, can prove infeasibility of scheduling attempt (often: fast)
Insights

How can an exact formulation be faster overall than the heuristic?

- ILP solver “sees” whole problem, can prove infeasibility of scheduling attempt (often: fast)
- Heuristic can only fail to find a solution in the given time budget
Insights

- Modulo SDC and Moovac complement each other
Insights

- Modulo SDC and Moovac complement each other
- “Synergistic scheduling”

Moovac: 489 min
Modulo SDC: 753 min
Combined: 429 min
Insights

- What makes Moovac better suited for HLS modulo scheduling than Eichenberger’s ILP?
What makes Moovac better suited for HLS modulo scheduling than Eichenberger’s ILP?

- Up to 1000+ operations, candidate IIs > 50 require humongous amounts of decision variables in time-indexed formulation
Insights

- What makes Moovac better suited for HLS modulo scheduling than Eichenberger’s ILP?
  - Up to 1000+ operations, candidate IIs > 50 require humongous amounts of decision variables in time-indexed formulation
  - Majority of ops is unconstrained, only subject to precedence constraints and exempt from all MRT handling in Moovac
Outlook

- Smarter search through the (rather large) II space
Outlook

- Smarter search through the (rather large) II space
  - Observation:
Smarter search through the (rather large) II space

- Observation:
- MaxII \( \rightarrow \) MinII ?

![Graph showing time to solution vs. candidate II]
- Smarter search through the (rather large) II space

  - Observation:
  - $\text{MaxII} \rightarrow \text{MinII}$?
  - Binary search?
Outlook

- Smarter search through the (rather large) II space
  - Observation:
    - MaxII $\rightarrow$ MinII?
  - Binary search?

- Integrate II search into the Moovac formulation
Smarter search through the (rather large) II space

- Observation:
- MaxII $\to$ MinII ?
- Binary search ?

Integrate II search into the Moovac formulation

- Time-indexed formulations:
  # decision variables dependent on candidate II
Conclusion

- Loop pipelining can reasonably be applied to wide range of HLS loops
Conclusion

- Loop pipelining can reasonably be applied to wide range of HLS loops
- The Modulo SDC heuristic delivers results on a par with exact formulations
## Conclusion

- Loop pipelining can reasonably be applied to wide range of HLS loops
- The Modulo SDC heuristic delivers results on a par with exact formulations
- The novel, exact Moovac formulation is surprisingly practical in its time-limited mode
Conclusion

- Loop pipelining can reasonably be applied to wide range of HLS loops
- The Modulo SDC heuristic delivers results on a par with exact formulations
- The novel, exact Moovac formulation is surprisingly practical in its time-limited mode
- Diverse options to reduce the scheduling time even further exist
Thank you!

oppermann@esa.tu-darmstadt.de