Contributions to Program Optimization and High-Level Synthesis
Habilitation Defense

Christophe Alias, Inria / LIP

Jury:
Corinne Ancourt, maître de recherche, MINES ParisTech, France
Isabelle Guérin-Lassous, professor, University of Lyon 1, France
Sebastian Hack, professor, University of Saarland, Germany
Michelle Strout, professor, University of Arizona, USA
Jürgen Teich, professor, Friedrich-Alexander University Erlangen-Nürnberg, Germany

May 15, 2019
Research topic: compilers for supercomputers

Automation of supercomputer design and programming
Target: scientific computing

Weather prediction

Chemical simulation

Astronomy
Target: scientific computing

- Weather prediction
  - Huge amount of computation
  - ... on petabytes of data
- Chemical simulation
- Astronomy
The trouble with power consumption

Power consumption is growing... under a limited power budget

⇒ energy efficiency (ops/W) is the performance measure
The trouble with power consumption

Power consumption is growing... under a limited power budget

⇒ energy efficiency (ops/W) is the performance measure

Trend: trade flexibility for energy efficiency
The trouble with power consumption

Power consumption is growing... under a limited power budget

⇒ energy efficiency (ops/W) is the performance measure

Trend: trade flexibility for energy efficiency

⇒ emergence of hardware accelerators (GPU, Xeon-Phi, FPGA)
⇒ heterogeneous supercomputers
FPGA: Field Programmable Gate Array = reconfigurable circuit
FPGA: Field Programmable Gate Array = reconfigurable circuit

Hope: better than GPUs (Microsoft CNN/FPGA: 53% more energy efficiency than GPU implementation).
FPGA: Field Programmable Gate Array = reconfigurable circuit

→ hope: better than GPUs (Microsoft CNN/FPGA: 53% more energy efficiency than GPU implementation).

High-level programming required!

→ need for robust, static automatic parallelization
Parallelization at a glance

- Task 1
- Task 2
- Task 3
- Task 4
- Task 5

- GPU
- FPGA

Our credo: rely on the compiler fine-grain automatic parallelization

Ad-hoc implementation

Library (e.g. BLAS)
Parallelization at a glance

- **Task 1**
- **Task 2**
- **Task 3**
- **Task 4**
- **Task 5**

**step 1**

**GPU**

**FPGA**

**step 2**

**step 3**
Parallelization at a glance

- **Task 1**
- **Task 2**
- **Task 3**
- **Task 4**
- **Task 5**

**step 1**

**step 2**

**step 3**

- **GPU**
- **FPGA**

Our credo: rely on the compiler fine-grain automatic parallelization.
Parallelization at a glance

Task 1

Task 2

Task 3

Task 4

Task 5

step 1

Cu

FPGA

step 2

Our credo: rely on the compiler fine-grain automatic parallelization

step 3

Library (e.g. BLAS)

Ad-hoc implementation
Parallelization at a glance

- **Task 1**
- **Task 2**
- **Task 3**
- **Task 4**
- **Task 5**

- **Step 1**
  - Task 1
  - Task 2

- **Step 2**
  - Task 3
  - Task 4

- **Step 3**
  - Task 5

Ad-hoc implementation
Library (e.g. BLAS)
Our credo: rely on the compiler
fine-grain automatic parallelization

- **GPU**
- **FPGA**
Automatic parallelization

for $i := 2$ to $N$
  for $j := 2$ to $N$
    $a[i,j] := a[i-1,j] + a[i,j-1]$;

Polyhedral model: the all-affine world
Automatic parallelization

\[
\begin{align*}
\text{for } i & := 2 \text{ to } N \\
\text{for } j & := 2 \text{ to } N \\
a[i,j] & := a[i-1,j] + a[i,j-1];
\end{align*}
\]

- **Polyhedral model**: the all-affine world
  \[
  \phi(i,j) = (i,j)
  \]
for $i := 2$ to $N$
  for $j := 2$ to $N$
    $a[i,j] := a[i-1,j] + a[i,j-1]$;

Polyhedral model: the all-affine world

$\phi(i, j) = (i, j)$  \hspace{1cm} $\theta(I, J, i, j) = (I + J, i, j)$
Automatic parallelization

\[
\begin{align*}
\text{for } i & := 2 \text{ to } N \\
\text{for } j & := 2 \text{ to } N \\
a[i,j] & := a[i-1,j] + a[i,j-1];
\end{align*}
\]

\textbf{Polyhedral model: the all-affine world}

\[
\begin{align*}
\phi(i, j) & = (i, j) \\
\theta(l, J, i, j) & = (l + J, i, j) \\
\Pi(l, J, i, j) & = J
\end{align*}
\]
Automatic parallelization

\begin{verbatim}
for i := 2 to N
  for j := 2 to N
    a[i,j] := a[i-1,j] + a[i,j-1];
\end{verbatim}

Polyhedral model: the all-affine world

\[ \phi(i, j) = (i, j) \quad \theta(I, J, i, j) = (I + J, i, j) \quad \Pi(I, J, i, j) = J \]
Automatic parallelization

\[ a[i, j] = a[i - 1, j] + a[i, j - 1] \quad \forall (i, j) \in D \]

- **Polyhedral model:** the all-affine world
  \[ \phi(i, j) = (i, j) \quad \theta(I, J, i, j) = (I + J, i, j) \quad \Pi(I, J, i, j) = J \]

- **Polyhedral representation:** System of affine recurrence equations (SARE) + Reductions (\( \sum_k \))
for $i := 2$ to $N$
    j := 2;
while $P$
    $a[i,j] := a[i-1,j] + a[u]$;
    j++;

\[ N = 5 \]
Automatic parallelization

\[
\text{for } i := 2 \text{ to } N
\]
\[
j := 2;
\]
\[
\text{while } P
\]
\[
a[i,j] := a[i-1,j] + a[u];
\]
\[
j++;
\]
Automatic parallelization

\[
\text{for } i := 2 \text{ to } N \\
j := 2; \\
\text{while } P \\
a[i,j] := a[i-1,j] + a[u]; \\
j++; \\
\]

- Sound abstraction $\leadsto$ polyhedral model [SAS10]
- Dynamic parallelization $\leadsto$ inspector/executor
- Mixed static/dynamic $\leadsto$ sparse polyhedral model
Contributions

Program transformations for automatic parallelization

Models and algorithms for high-level synthesis
Contributions

Program transformations for automatic parallelization

- Monoparametric tiling [IMPACT14], semantic tiling [LASHC13]
- Algorithm recognition [SAS14, IoossPhD16]

⇝ Guillaume Iooss’ PhD

Models and algorithms for high-level synthesis

Communication synthesis [ASAP10, PPoPP12, IMPACT12, DATE13] 
⇝ Alexandru Plesco’s PhD

Data-aware process networks [Patent14]

Control scheduling [ARC11, MICPRO12] and synthesis [TACO17]

Channel typing [HiP3ES18], allocation/sizing [LCTES07], synchronization [Patent14]

⇒ Industrial transfer: XtremLogic

Software transferred (Inria licence):
⇝ dcc (5556 loc C++, 7 hm) and poco (20054 loc, 28 hm)

Scientific advising
Program transformations for automatic parallelization

- Monoparametric tiling [IMPACT14], semantic tiling [LASHC13]
- Algorithm recognition [SAS14, IoossPhD16]

→ Guillaume Iooss’ PhD

Models and algorithms for high-level synthesis

- Communication synthesis [ASAP10, PPoPP12, IMPACT12, DATE13]

→ Alexandru Plesco’s PhD
Contributions

Program transformations for automatic parallelization
- Monoparametric tiling [IMPACT14], semantic tiling [LASHC13]
- Algorithm recognition [SAS14, IoossPhD16]

⇾ Guillaume Iooss’ PhD

Models and algorithms for high-level synthesis
- Communication synthesis [ASAP10, PPoPP12, IMPACT12, DATE13]
  ⇾ Alexandru Plesco’s PhD
- Data-aware process networks [Patent14]
- Control scheduling [ARC11, MICPRO12] and synthesis [TACO17]
- Channel typing [HiP3ES18], allocation/sizing [LCTES07], synchronization [Patent14]

⇒ Industrial transfer: XtremLogic
  - Software transferred (Inria licence):
    ⇾ dcc (5556 loc C++, 7 hm) and poco (20054 loc, 28 hm)
  - Scientific advising
Contributions

Program transformations for automatic parallelization
- Monoparametric tiling [IMPACT14], semantic tiling [LASHC13]

Algorithm recognition [SAS14, IoossPhD16]
⇝ Guillaume Iooss’ PhD

Models and algorithms for high-level synthesis
- Communication synthesis [ASAP10, PPoPP12, IMPACT12, DATE13]
  ⇝ Alexandru Plesco’s PhD
- Data-aware process networks [Patent14]
- Control scheduling [ARC11, MICPRO12] and synthesis [TACO17]
- Channel typing [HiP3ES18], allocation/sizing [LCTES07], synchronization [Patent14]
⇒ Industrial transfer: XtremLogic
  - Software transferred (Inria licence):
    ⇝ dcc (5556 loc C++, 7 hm) and poco (20054 loc, 28 hm)
  - Scientific advising
1 Introduction

2 Focus: Algorithm recognition

3 Focus: High-level synthesis for FPGA

4 Conclusion and perspectives
Focus: algorithm recognition

Goal

**Automatic refactoring** with a performance library (e.g. BLAS)

Program

Library
Focus: algorithm recognition

Goal

Automatic refactoring with a performance library (e.g. BLAS)

Program

- trmm
- syrk
- gemm
- trsm

Library

Refactoring
Focus: algorithm recognition

Goal

**Automatic refactoring** with a performance library (e.g. BLAS)

Challenges:

- Program equivalence is **undecidable** on polyhedral programs
- Deal with **reductions** ($\sum_k$, key operation)
  - Recognition modulo associativity & commutativity
  - Other semantic properties in linear algebra
- Select the **best refactoring** $\leadsto$ performance prediction
Partition the computation to expose “submatrices” and operations on submatrices
Partition the computation to expose “submatrices” and operations on submatrices

Tiling: group computation into atomic tiles
Partition the computation to expose “submatrices” and operations on submatrices

Tiling: group computation into atomic tiles

\[ \text{monoparametric tiling} \ [\text{IMPACT14}] \]
**Partition** the computation to expose “submatrices” and operations on submatrices

**Tiling:** group computation into atomic tiles

\[ \text{monoparametric tiling } [\text{IMPACT14}] \]
Approach

Semantic tiling: group data into rectangular tiles [LASHC13, IoossPhD16]

- Rectangular data tiles = submatrices
- Induced computation tiles = operations on submatrices
**Semantic tiling:** group data into rectangular tiles [LASHC13,IoossPhD16]

- Rectangular data tiles = submatrices
- Induced computation tiles = operations on submatrices
**Semantic tiling:** group data into rectangular tiles [LASHC13, IoossPhD16]

- Rectangular data tiles = submatrices
- Induced computation tiles = operations on submatrices
Contributions

Original Program

Semantic tiling [IMPACT12, LASHC13, IoossPhD16]

Main System

Subsystems
Contributions

Original Program

Semantic tiling [IMPACT12, LASHC13, IoossPhD16]

Main System

Subsystems

Template Matching [SAS14, IoossPhD16]

Template Library

Match

No match

Template Recognized

Unknown Computation
Equivalence checking [Barthou01]

\[ O = A[N] \]
\[ A[i] = \begin{cases} 
  i > 0 & : f(A[i - 1], I[i]) \\
  i = 0 & : I[0]
\end{cases} \]

\[ O' = f(A'[N - 1], I[N]) \]
\[ A'[i] = \begin{cases} 
  i > 0 & : f(A'[i - 1], I[i]) \\
  i = 0 & : I[0]
\end{cases} \]

Normalize and check the terms \(\leadsto\) Herbrand equivalence

Unrolling + comparison: integer interpreted automaton
\[ O = A[N] \]
\[ A[i] = \begin{cases} 
  i > 0 &: f(A[i - 1], I[i]) \\
  i = 0 &: I[0] 
\end{cases} \]

\[ O' = f(A'[N - 1], I[N]) \]
\[ A'[i] = \begin{cases} 
  i > 0 &: f(A'[i - 1], I[i]) \\
  i = 0 &: I[0] 
\end{cases} \]

- Normalize and check the terms \( \rightsquigarrow \) Herbrand equivalence
- Unrolling + comparison: integer interpreted automaton
**Equivalence checking [Barthou01]**

\[ O = A[N] \]

\[ A[i] = \begin{cases} 
  i > 0 : & f(A[i-1], I[i]) \\
  i = 0 : & f(I[0]) \
\end{cases} \]

\[ O' = f(A'[N-1], I[N]) \]

\[ A'[i] = \begin{cases} 
  i > 0 : & f(A'[i-1], I[i]) \\
  i = 0 : & f(I[0]) \
\end{cases} \]

- Normalize and check the terms \( \rightsquigarrow \text{Herbrand equivalence} \)
- Unrolling + comparison: integer interpreted automaton
From equivalence to recognition [ IoossPhD16 ]

Library algorithms

Semantic tile
Library algorithms

Semantic tile

From equivalence to recognition [IoossPhD16]
Approach

- View an algorithm as an input-parametrized template
- Match the algorithm to the program, iterate on the unifiers
Approach

- View an algorithm as an input-parametrized template
- Match the algorithm to the program, iterate on the unifiers
From equivalence to recognition [loossPhD16]

Library algorithms

Semantic tile

Approach

- View an algorithm as an input-parametrized template
- Match the algorithm to the program, iterate on the unifiers
**Approach**

- View an algorithm as an input-parametrized **template**
- **Match** the algorithm to the program, iterate on the **unifiers**
Experimental results

**Platform:** Intel Xeon CPU E5-1650 (Freq: 3.50GHz)

**Operating System:** Fedora 23

**Library:** BLAS subset levels 1,2,3

**Applications:**
- Symmetric Positive semi-Definite Matrix Inversion (SPDMI)
- Sylvester Equation Solver \((A.X + X.B = C)\)
- Algebraic Path Problem (APP / Floyd-Warshall for \((\text{max, +})\))
- McCaskill (bioinformatics application for RNA analysis)
Experimental results

Platform: Intel Xeon CPU E5-1650 (Freq: 3.50GHz)
Operating System: Fedora 23
Library: BLAS subset levels 1,2,3

Applications:
- Symmetric Positive semi-Definite Matrix Inversion (SPDMI)
- Sylvester Equation Solver \((A.X + X.B = C)\)
- Algebraic Path Problem (APP / Floyd-Warshall for \((\text{max,+})\))
- McCaskill (bioinformatics application for RNA analysis)

Results:

<table>
<thead>
<tr>
<th>Application</th>
<th>SPDMI</th>
<th>Sylvester</th>
<th>APP</th>
<th>McCaskill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsystems</td>
<td>16</td>
<td>8</td>
<td>60</td>
<td>113</td>
</tr>
<tr>
<td>#Fully recognized</td>
<td>12</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Templates detected</td>
<td>27</td>
<td>8</td>
<td>44</td>
<td>80</td>
</tr>
<tr>
<td>Missing part</td>
<td>(O(n^2)/O(n^3))</td>
<td>(O(n^2)/O(n^3))</td>
<td>(O(n^2)/O(n^3))</td>
<td>almost all</td>
</tr>
<tr>
<td>Time/matching</td>
<td>2 s</td>
<td>1.5 s</td>
<td>2.3 s</td>
<td>1.9 s</td>
</tr>
<tr>
<td>Total time</td>
<td>8 min</td>
<td>2 min</td>
<td>27 min</td>
<td>1 h 10 min</td>
</tr>
</tbody>
</table>
Conclusion on this part

A complete system for algorithm recognition:
- Based on semantic tiling and template matching
- Application: program refactoring with a performance library

Future work:
- Find a performance model to choose a relevant refactoring
- Explore other applications for semantic tiling
- Explain the fractality of semantic tiling
1 Introduction
2 Focus: Algorithm recognition
3 Focus: High-level synthesis for FPGA
4 Conclusion and perspectives
**Goal**

Models and algorithms towards complete polyhedral-powered HLS for FPGA

→ XtremLogic start-up

```c
void kernel_cholesky(double **A, int N)
{
    int i, j, k;
    for (i = 0; i < N; i++) {
        for (j = 0; j < i; j++) {
            for (k = 0; k < j; k++)
        }
        for (k = 0; k < i; k++)
        A[i][i] = sqrt(A[i][i]);
    }
}
```
Focus: High-level synthesis for FPGA

Goal

Models and algorithms towards complete polyhedral-powered HLS for FPGA

→ XtremLogic start-up

```c
void kernel_cholesky(double **A, int N)
{
    int i, j, k;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < i; j++)
        {
            for (k = 0; k < j; k++)
            A[i][i] = A[i][i] / A[i][i];
        }
        for (k = 0; k < i; k++)
    }
    A[i][i] = sqrt(A[i][i]);
}
```
Focus: High-level synthesis for FPGA

**Goal**

Models and algorithms towards complete polyhedral-powered HLS for FPGA

→ XtremLogic start-up
Goal

Models and algorithms towards complete polyhedral-powered HLS for FPGA

→ XtremLogic start-up

Challenges:

- **Orchestrate data spilling to the DDR**
  - Limited local storage (8 Mbytes on Stratix 10 GX1150)
  - Applications with large storage footprint

- Allow tunable operational intensity and parallelism

- Express pipelined parallelism (circuit)
Approach

Data-aware process network (DPN)

- Produce a dataflow model with explicit data spilling
- Tunable local storage / operational intensity and parallelism
**Approach**

Data-aware process network (DPN)

- Produce a dataflow model with explicit data spilling
- Tunable local storage / operational intensity and parallelism
- Synthesize the dataflow model to FPGA (XtremLogic)
Data-aware process networks (DPN) [Patent14]
Contributions

Data-aware process networks (DPN) [Patent14]

Communication synthesis
[ASAP10, PPoPP12, IMPACT12, DATE13]
Contributions

Data-aware process networks (DPN) [Patent14]

Communication synthesis [ASAP10, PPoPP12, IMPACT12, DATE13]

- Control scheduling [ARC11, MICPRO12]
- Control synthesis [TACO17]
Contributions

Data-aware process networks (DPN) [Patent14]

Communication synthesis
[ASAP10, PPoPP12, IMPACT12, DATE13]

- Control scheduling [ARC11, MICPRO12]
- Control synthesis [TACO17]

- Channel typing [HiP3ES18]
- Channel allocation/sizing [LCTES07]
- Synchronizations [Patent14]
Contributions

Data-aware process networks (DPN) \[\text{Patent14}\]

Communication synthesis
\[\text{ASAP10,PPoPP12,IMPACT12,DATE13}\]
- Control scheduling \[\text{ARC11,MICPRO12}\]
- Control synthesis \[\text{TACO17}\]
- Channel typing \[\text{HiP3ES18}\]
- Channel allocation/sizing \[\text{LCTES07}\]
- Synchronizations \[\text{Patent14}\]

Software: \text{dcc (DPN C Compiler)}
Transferred to XtremLogic under an Inria license
Contributions

Data-aware process networks (DPN) \[\text{[Patent14]}\]

Communication synthesis
\[\text{[ASAP10, PPoPP12, IMPACT12, DATE13]}\]
- Control scheduling \[\text{[ARC11, MICPRO12]}\]
- Control synthesis \[\text{[TAC017]}\]
- Channel typing \[\text{[HiP3ES18]}\]
- Channel allocation/sizing \[\text{[LCTES07]}\]
- Synchronizations \[\text{[Patent14]}\]

Software: dcc (DPN C Compiler)
Transferred to XtremLogic under an Inria license
Regular process networks

\[
\begin{align*}
&\text{for } i := 0 \text{ to } N \\
&\quad \bullet \ a[i] = f(i); \\
&\text{for } i := 1 \text{ to } N \\
&\quad \bullet \ b[i] := a[i - 1] + a[i];
\end{align*}
\]

\[\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & i \\
\end{array}\]

- Partition of the computation: processes
- Partition of \(\rightarrow_{pc}\): channels \(\{\rightarrow_1, \rightarrow_2, \ldots\}\)
- A schedule \(\theta_P\) for each process \(P\)
Regular process networks

for $i := 0$ to $N$

- $a[i] = f(i)$;

for $i := 1$ to $N$

- $b[i] := a[i - 1] + a[i]$;

- Partition of the computation: processes
- Partition of $\rightarrow_{pc}$: channels $\{\rightarrow_1, \rightarrow_2, \ldots\}$
- A schedule $\theta_P$ for each process $P$
Regular process networks

\[
\begin{align*}
\text{for } i & := 0 \text{ to } N \\
\bullet & \ a[i] = f(i); \\
\text{for } i & := 1 \text{ to } N \\
\bullet & \ b[i] := a[i - 1] + a[i];
\end{align*}
\]

\[\begin{array}{c}
P_1 \\
P_2 \\
P_3 \\
P_4
\end{array}\]

- Partition of the computation: processes
- Partition of $\rightarrow_{pc}$: channels $\{\rightarrow_1, \rightarrow_2, \ldots\}$
- A schedule $\theta_P$ for each process $P$
Regular process networks

\[\text{for } i := 0 \text{ to } N\]
- \(a[i] = f(i)\);

\[\text{for } i := 1 \text{ to } N\]
- \(b[i] := a[i - 1] + a[i]\);

- Partition of the computation: processes
- Partition of \(\rightarrow_{pc}\): channels \(\{\rightarrow_1, \rightarrow_2, \ldots\}\)
- A schedule \(\theta_P\) for each process \(P\)
Regular process networks

for $i := 0$ to $N$
• $a[i] = f(i)$;
for $i := 1$ to $N$
• $b[i] := a[i - 1] + a[i]$;

Locally sequential
Globally dataflow

Partition of the computation: processes
Partition of $\rightarrow_{pc}$: channels \{$\rightarrow_1, \rightarrow_2, \ldots$\}
A schedule $\theta_P$ for each process $P$
HLS methodology

1. Setup the RPN partitioning strategy (e.g. PPN, DPN)
2. Front-end: kernel → RPN

Benefits:
- Combines the benefits of partitioning and dataflow models
- Explicit and typed communications (FIFO, buffer, DDR)

```
for i := 0 to N
  • a[i] = f(i);
for i := 1 to N
  • b[i] := a[i - 1] + a[i];
```
Data-aware process networks [Patent14]

for \( t := 1 \) to \( T \)

for \( i := 1 \) to \( N - 2 \)

\[
a[t, i] := a[t - 1, i - 1] + a[t - 1, i] + a[t - 1, i + 1];
\]

Tiling \( \phi_S(t, i) = (t, t + i) \)

- **Load/store:** consider tile bands as reuse units [ASAP10,DATE13]
- Pipelined comm.: Load(\( T \)) → C(\( T \)) → Store(\( T \))
for \( t := 1 \) to \( T \)
for \( i := 1 \) to \( N - 2 \)
\[
a[t, i] := a[t - 1, i - 1] + a[t - 1, i] + a[t - 1, i + 1];
\]

Tiling \( \phi_S(t, i) = (t, t + i) \)

- **Load/store**: consider tile bands as reuse units [ASAP10, DATE13]
- **Pipelined comm.**: Load\((T) \rightarrow C(T) \rightarrow \text{Store}(T)\)
for $t := 1$ to $T$
for $i := 1$ to $N - 2$
    $a[t, i] := a[t - 1, i - 1] + a[t - 1, i] + a[t - 1, i + 1]$;

Tiling $\phi_S(t, i) = (t, t + i)$

Load/store: consider tile bands as reuse units [ASAP10,DATE13]

$\leadsto$ Pipelined comm.: Load($T$) $\rightarrow$ C($T$) $\rightarrow$ Store($T$)
for $t := 1$ to $T$
for $i := 1$ to $N - 2$
\[ a[t, i] := a[t - 1, i - 1] + a[t - 1, i] + a[t - 1, i + 1]; \]

Tiling $\phi_S(t, i) = (t, t + i)$

Load/store: consider tile bands as reuse units [ASAP10,DATE13]

$\leadsto$ Pipelined comm.: $\text{Load}(T) \rightarrow C(T) \rightarrow \text{Store}(T)$

Parallelism: split tile band with outer tiling hyperplanes (ex: $t$)
Data-aware process networks (DPN) [Patent14]

- Communication synthesis [ASAP10, PPoPP12, IMPACT12, DATE13]
- Control scheduling [ARC11, MICPRO12]
- Control synthesis [TAC017]
- Channel typing [HiP3ES18]
- Channel allocation/sizing [LCTES07]
- Synchronizations [Patent14]

Software: dcc (DPN C Compiler)
Transferred to XtremLogic under an Inria license
Finite state machine

First $\mathcal{P} = \{ T \geq 1 \land N \geq 3 : (0, 0, 1, 1) \}

Next $\mathcal{P}(l_1, l_2, t, i) = \left\{ \begin{array}{l}
(d\text{epth } l_1) \\
-4 + T - 4l_1 \geq 0 : (1 + l_1, 1 + l_1, 4 + 4l_1, 1) \\

(d\text{epth } l_2) \\
-3 + N + 4l_1 - 4l_2 \geq 0 \land 6 - N - 4l_1 + 4l_2 \geq 0 \land \\
-6 + N + T - 4l_2 \geq 0 \land -5 + N - 4l_2 < 0 : (h_1, h_2, 6 - N + 4l_2, -2 + N) \\

l_1 \geq 0 \land -5 + N - 4l_2 \geq 0 \\
(h_1, 1 + l_2, 4l_1, 4 - 4l_1 + 4l_2) \\

6 - N - 4l_1 + 4l_2 < 0 \land -6 + N + T - 4l_2 \geq 0 \land -5 + N - 4l_2 < 0 \\
(h_1, 1 + l_2, 4l_1, 4 - 4l_1 + 4l_2) \\

-5 + N - 4l_2 \geq 0 \land -5 + N - 4l_2 \geq 0 \\
(h_1, 1 + l_2, 1, 3 + 4l_2) \\

(d\text{epth } t) \\
2 + t - 4l_2 \geq 0 \land 1 - t + 4l_2 \geq 0 \land \\
2 - t + 4l_1 \geq 0 \land -1 + T - t \geq 0 \\
(h_1, h_2, 1 + t, 1) \\

2 + t - 4l_2 < 0 \land 1 - t + 4l_2 \geq 0 \land \\
2 - t + 4l_1 \geq 0 \land -1 + T - t \geq 0 \\
(h_1, h_2, 1 + t, -1 - t + 4l_2) \\

(d\text{epth } i) \\
-3 + N - i \geq 0 \land 2 - t - i + 4l_2 \geq 0 \\
(h_1, h_2, t, 1 + i)
\end{array} \right\}
Steering logic

\[ \text{mux}(\langle C, l_1, l_2, t, i \rangle, 1) = \]
\[ \begin{cases} 
  t = 4l_1 : & \text{buffer1}[i - 1] \\
  t > 4l_1 \land i = 1 : & \text{buffer1}[i - 1] \\
  t > 4l_1 \land i > 1 : & \text{buffer4}[t - 1, i - 1] 
\end{cases} \]

\[ \text{demux}(\langle C, l_1, l_2, t, i \rangle) = \]
\[ \begin{cases} 
  i \leq N - 3 \land t < 4l_1 + 3 \land t \leq T - 1 : & \text{buffer4}[t, i] \\
  i \leq N - 2 \land t < 4l_1 + 3 \land t \leq T - 1 : & \text{buffer5}[t, i] \\
  i \leq N - 2 \land t < 4l_1 + 3 \land t \leq T - 1 : & \text{buffer6}[t, i] \\
  t = 4l_1 + 3 : & \text{buffer13}[i] 
\end{cases} \]
Compact the affine expressions/constraints to an hardware-efficient DAG

Challenges:
- Volume of affine expressions and constraints
- Must be evaluated in parallel
- Common subexpression factorization is not sufficient
Expression factorization

\[ E_1 = i + 2j + k \]
\[ E_2 = 5i + 2j + 3k \]

\[ \begin{array}{c}
E_1 \\
\downarrow \\
+ \\
5i + 3k \\
\end{array} \quad \begin{array}{c}
E_2 \\
\downarrow \\
+ \\
2j \\
\end{array} \quad \begin{array}{c}
+ \\
i + k \\
\end{array} \quad \begin{array}{c}
5i + 2j + 3k \\
\end{array} \]

<table>
<thead>
<tr>
<th>CSE</th>
<th>+</th>
<th>\times \text{constant}</th>
<th>shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Expression factorization

\[ E_1 = i + 2j + k \]
\[ E_2 = 5i + 2j + 3k \]

remark that \( E_2 = E_1 + 4i + 2k \)

\[ \begin{array}{c|c|c|c}
\text{E1} & \text{E2} & \text{CSE} \\
+ & + & \\
5i + 3k & 2j & i + k \\
\hline
\end{array} \]
Expression factorization

\[ E_1 = i + 2j + k \]
\[ E_2 = 5i + 2j + 3k \]

remark that \( E_2 = E_1 + 4i + 2k \)

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>× constant</th>
<th>shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSE</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>OPT</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
Constraint factorization

\( C_1 : 4i + 3j < 0 \)
\( C_2 : -5i - 3j - 1 < 0 \)

\[
\begin{align*}
C_1 & : 4i + 3j < 0 \\
C_2 & : -5i - 3j - 1 < 0 \\
\end{align*}
\]

\[
\begin{array}{c|c|c|c}
 & + & \times \text{ constant} & \text{shift} \\
\hline
\text{CSE} & 4 & 3 & 1 \\
\end{array}
\]
Constraint factorization

\[ C_1 : 4i + 3j < 0 \]
\[ C_2 : -5i - 3j - 1 < 0 \]

remark that: \[ C_2 : 5i + 3j \geq 0 \]

\[
\begin{array}{c|c}
C_1 & C_2 \\
\hline
< 0 & < 0 \\
4i + 3j & -5i - 3j - 1 \\
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
 & + & \times \text{ constant} & \text{shift} \\
\hline
\text{CSE} & 4 & 3 & 1 \\
\hline
\end{array}
\]
Constraint factorization

\[ C_1 : 4i + 3j < 0 \]
\[ C_2 : -5i - 3j - 1 < 0 \]

Remark that:
\[ C_2 : 5i + 3j \geq 0 \]

\[
\begin{array}{c|c}
\text{CSE} & 4 & 3 & 1 \\
\text{OPT} & 2 & 1 & 1 \\
\end{array}
\]
Nodes:
expressions, constraints, CSE

Edges: $u \xrightarrow{\Delta} v$:
$v$ realized from $u$ at cost $\Delta$

<table>
<thead>
<tr>
<th>Id</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$i + 2j + k &lt; 0$</td>
</tr>
<tr>
<td>2</td>
<td>$5i + 2j + 3k &lt; 0$</td>
</tr>
<tr>
<td>3</td>
<td>$4i + 3j &lt; 0$</td>
</tr>
<tr>
<td>4</td>
<td>$-5i - 3j - 1 &lt; 0$</td>
</tr>
</tbody>
</table>
Realization graph

Nodes:
expressions, constraints, CSE

Edges:  \( u \xrightarrow{\Delta} v \):
v realized from \( u \) at cost \( \Delta \)

Realization of \( v \):
initial_node \( \xrightarrow{\Delta_1} u_1 \ldots \xrightarrow{\Delta_n} u_n \xrightarrow{\Delta} v \)

Global realization:
spanning tree

<table>
<thead>
<tr>
<th>Id</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( i + 2j + k &lt; 0 )</td>
</tr>
<tr>
<td>2</td>
<td>( 5i + 2j + 3k &lt; 0 )</td>
</tr>
<tr>
<td>3</td>
<td>( 4i + 3j &lt; 0 )</td>
</tr>
<tr>
<td>4</td>
<td>( -5i - 3j - 1 &lt; 0 )</td>
</tr>
</tbody>
</table>
Realization graph

Nodes:
expressions, constraints, CSE

Edges: $u \xrightarrow{\Delta} v$:
$v$ realized from $u$ at cost $\Delta$

Realization of $v$:
$initial\_node \xrightarrow{\Delta_1} u_1 \ldots \xrightarrow{\Delta_n} u_n \xrightarrow{\Delta} v$

Global realization:
spanning tree

Best realization:
minimum spanning tree
Experimental setup

**Kernels:** PolyBench/C v3.2

**Target:** FPGA Arria 10 10AX115S2F4I1SG

**Synthesis:** Intel Quartus Prime TM 16.1.2

**Methodology:**
- Control synthesis per process
- Compare:
  - **SEM+Quartus:** Our DAG, pipelined
  - **Quartus:** Direct implementation (affine control in VHDL) + output registers
### Experimental results

<table>
<thead>
<tr>
<th>Kernel</th>
<th>#dags</th>
<th>#C</th>
<th>#E</th>
<th>SEM+Quartus ALM</th>
<th>SEM+Quartus Regs</th>
<th>Quartus ALM</th>
<th>Quartus Regs</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>2mm</td>
<td>15</td>
<td>250</td>
<td>161</td>
<td>1011</td>
<td>612</td>
<td>1430</td>
<td>596</td>
<td>29%</td>
</tr>
<tr>
<td>3mm</td>
<td>18</td>
<td>369</td>
<td>206</td>
<td>1775</td>
<td>946</td>
<td>2528</td>
<td>922</td>
<td>30%</td>
</tr>
<tr>
<td>atax</td>
<td>12</td>
<td>134</td>
<td>83</td>
<td>628</td>
<td>328</td>
<td>900</td>
<td>321</td>
<td>30%</td>
</tr>
<tr>
<td>bicg</td>
<td>11</td>
<td>112</td>
<td>69</td>
<td>500</td>
<td>278</td>
<td>715</td>
<td>278</td>
<td>30%</td>
</tr>
<tr>
<td>correlation</td>
<td>27</td>
<td>356</td>
<td>205</td>
<td>1609</td>
<td>1129</td>
<td>2567</td>
<td>909</td>
<td>37%</td>
</tr>
<tr>
<td>covariance</td>
<td>16</td>
<td>243</td>
<td>143</td>
<td>1221</td>
<td>708</td>
<td>1872</td>
<td>618</td>
<td>35%</td>
</tr>
<tr>
<td>doitgen</td>
<td>9</td>
<td>145</td>
<td>124</td>
<td>393</td>
<td>280</td>
<td>607</td>
<td>268</td>
<td>35%</td>
</tr>
<tr>
<td>fdtd-2d</td>
<td>13</td>
<td>502</td>
<td>167</td>
<td>2339</td>
<td>1713</td>
<td>3293</td>
<td>1603</td>
<td>29%</td>
</tr>
<tr>
<td>gemm</td>
<td>10</td>
<td>125</td>
<td>93</td>
<td>672</td>
<td>270</td>
<td>851</td>
<td>270</td>
<td>21%</td>
</tr>
<tr>
<td>gemver</td>
<td>20</td>
<td>187</td>
<td>137</td>
<td>847</td>
<td>459</td>
<td>1102</td>
<td>438</td>
<td>23%</td>
</tr>
<tr>
<td>gesummv</td>
<td>14</td>
<td>95</td>
<td>84</td>
<td>456</td>
<td>245</td>
<td>549</td>
<td>227</td>
<td><strong>17%</strong></td>
</tr>
<tr>
<td>heat-3d</td>
<td>8</td>
<td>734</td>
<td>175</td>
<td>3545</td>
<td>1194</td>
<td>5667</td>
<td>2559</td>
<td>37%</td>
</tr>
<tr>
<td>jacobi-1d</td>
<td>8</td>
<td>134</td>
<td>64</td>
<td>628</td>
<td>556</td>
<td>912</td>
<td>520</td>
<td>31%</td>
</tr>
<tr>
<td>jacobi-2d</td>
<td>8</td>
<td>370</td>
<td>111</td>
<td>1660</td>
<td>1204</td>
<td>2547</td>
<td>1144</td>
<td>35%</td>
</tr>
<tr>
<td>lu</td>
<td>7</td>
<td>213</td>
<td>87</td>
<td>1116</td>
<td>666</td>
<td>1469</td>
<td>628</td>
<td>24%</td>
</tr>
<tr>
<td>mvt</td>
<td>11</td>
<td>118</td>
<td>70</td>
<td>550</td>
<td>290</td>
<td>758</td>
<td>290</td>
<td>27%</td>
</tr>
<tr>
<td>seidel-2d</td>
<td>5</td>
<td>226</td>
<td>63</td>
<td>1161</td>
<td>1464</td>
<td>1758</td>
<td>1291</td>
<td>34%</td>
</tr>
<tr>
<td>symm</td>
<td>13</td>
<td>213</td>
<td>116</td>
<td>1011</td>
<td>471</td>
<td>1540</td>
<td>465</td>
<td>34%</td>
</tr>
<tr>
<td>syr2k</td>
<td>10</td>
<td>135</td>
<td>90</td>
<td>721</td>
<td>290</td>
<td>944</td>
<td>281</td>
<td>24%</td>
</tr>
<tr>
<td>syrk</td>
<td>9</td>
<td>118</td>
<td>81</td>
<td>636</td>
<td>246</td>
<td>828</td>
<td>246</td>
<td>23%</td>
</tr>
<tr>
<td>trisolv</td>
<td>9</td>
<td>93</td>
<td>56</td>
<td>474</td>
<td>218</td>
<td>632</td>
<td>213</td>
<td>25%</td>
</tr>
<tr>
<td>trmm</td>
<td>8</td>
<td>118</td>
<td>79</td>
<td>549</td>
<td>262</td>
<td>806</td>
<td>253</td>
<td>32%</td>
</tr>
</tbody>
</table>

**Maximum FPGA frequency (645 MHz), High compression ratio (5-6 input LUT)**

**gesummv**: small bitwidth → low-level boolean optimizations are more effective
Conclusion on this part

Models and algorithms for polyhedral HLS:

- **DPN**, a dataflow intermediate representation cross fertilizing dataflow models and partitioning
- Benefits: explicit data spilling, natural tuning of arithmetic intensity and parallelism
- Front-end and back-end compiler algorithms from C programs
  - ⇒ Software: dcc, transferred to XtremLogic

**Future work:**

- DPN-level resource factorization (control/channels)
- Scheduling for latency minimization (pipelined datapath)
Outline

1. Introduction
2. Focus: Algorithm recognition
3. Focus: High-level synthesis for FPGA
4. Conclusion and perspectives
Compiler algorithms to generate efficient software and hardware for HPC kernels → Industrial transfer: XtremLogic

Deeply influenced by the polyhedral model, and the advances in the last decade.

Major concepts:
- (affine) tiling, the key transformation
- dataflow models, the key representation

Major locks:
- space/time scalability
- fork/join nature of polyhedral scheduling
Partial compilation

- Let parameters (parallelism, local footprint) survive the compilation
- **Application**: HLS/FPGA: tune the parallelism of a circuit
- **Challenges**: How to parametrize a DPN? What would be a generic parallel process?
Partial compilation

- Let parameters (parallelism, local footprint) survive the compilation
- **Application:** HLS/FPGA: tune the parallelism of a circuit
- **Challenges:** How to parametrize a DPN? What would be a generic parallel process?

Lazy compilation

- **Complexity:** set subtraction, min/max of piecewise affine mappings
- **Idea:** hide complexity with lazy values, evaluated dynamically (e.g. \( R := P \setminus Q \))
- **Challenges:** How to compose/simplify lazy values? How to rephrase compiler analysis with lazy values?
Thanks to

Co-authors:

Fabrice Baray, Denis Barthou, Uday Bondhugula, Yongjian Chen, Alain Darte, Paul Feautrier, Carsten Fuhs, Laure Gonnord, Guobin He, Thomas Henretty, Guillaume Iooss, Sriram Krishnamoorthy, Haibo Lin, Qingda Lu, Tin-fook Ngai, Bogdan Pasca, Alexandru Plesco, Sanjay Rajopadhye, J. Ramanujam, Fabrice Rastello, Lawrence Rauchwerger, Atanas Rountev, Silvius Rus, P. Sadayappan, Yun Zou

PhD students:

Guillaume Iooss, Alexandru Plesco
\[
O = A[N]
\]
\[
A[i] = \begin{cases} 
  i > 0 & : f(A[i - 1], I[i]) \\
  i = 0 & : I[0]
\end{cases}
\]

\[
O = f(A'[N - 1], I[N])
\]
\[
A'[i] = \begin{cases} 
  i > 0 & : f(A'[i - 1], I[i]) \\
  i = 0 & : I[0]
\end{cases}
\]

Precondition: \(N > 0\)
Equivalence checking [Barthou01]

\[ O = O' \]

\[ O = A[N] \]
\[ A[i] = \begin{cases} 
  i > 0 : & f(A[i-1], I[i]) \\
  i = 0 : & I[0]
\end{cases} \]

\[ O = f(A'[N-1], I[N]) \]
\[ A'[i] = \begin{cases} 
  i > 0 : & f(A'[i-1], I[i]) \\
  i = 0 : & I[0]
\end{cases} \]

Precondition: \( N > 0 \)
Equivalence checking [Barthou01]

O = A[N]
A[i] = \begin{cases} 
i > 0 & : f(A[i-1], I[i]) 
i = 0 & : I[0] \end{cases}

Precondition: \( N > 0 \)
Equivalence checking [Barthou01]

\[ O = O' \]
\[ A[N] = f(A'[N - 1], I[N]) \] (Gen) \( i = N \)
\[ A[i] = f(A'[N - 1], I[N]) \]
\[ i = 0 \] (Comp)
\[ I[0] = f(A'[N - 1], I[N]) \]

failure if \( N = 0 \)

\[ O = A[N] \]
\[ A[i] = \begin{cases} 
  i > 0 & : f(A[i - 1], I[i]) \\
  i = 0 & : I[0]
\end{cases} \]
\[ i = 0 \] (Comp)
\[ I[0] = f(A'[N - 1], I[N]) \]

Precondition: \( N > 0 \)

\[ O = f(A'[N - 1], I[N]) \]
\[ A'[i] = \begin{cases} 
  i > 0 & : f(A'[i - 1], I[i]) \\
  i = 0 & : I[0]
\end{cases} \]
Equivalence checking [Barthou01]

\[
\begin{align*}
O &= A[N] \\
A[i] &= \begin{cases} 
  i > 0 & : f(A[i-1], I[i]) \\
  i = 0 & : I[0]
\end{cases} \\
A'[i] &= \begin{cases} 
  i > 0 & : f(A'[i-1], I[i]) \\
  i = 0 & : I[0]
\end{cases}
\end{align*}
\]

Precondition: \( N > 0 \)

\[
O = A[N] \\
A[i] = f(A'[N - 1], I[N]) \\
i = 0 \\
(Comp)
\]

\[
A[N] = f(A'[N - 1], I[N]) \\
i = N \\
(Gen)
\]

\[
A[i] = f(A'[N - 1], I[N]) \\
i > 0
\]

\[
I[0] = f(A'[N - 1], I[N]) \\
f(A[i - 1], I[i]) = f(A'[N - 1], I[N])
\]

\[
(Dec)
\]

\[
I[i] = I[N] \\
A[i - 1] = A'[N - 1] \\
i = i - 1 \\
(Gen)
\]

\[
i = i - 1 \\
i' = i' - 1
\]

\[
i > 0, i' > 0 \\
(Dec)
\]

\[
i > 0, i' > 0 \\
f(A[i - 1], I[i]) = f(A'[i' - 1], I[i'])
\]

\[
(Comp)
\]

\[
I[0] = I[0] \\
\vdots \\
\vdots
\]

\[
I[i] = I[i']
\]

Success if \( i = i' \)

Failure states are unreachable

Success states \( I[i] = I[i'] \) are reached with \( i = i' \)
Equivalence checking [Barthou01]

\[ O = O' \]

(Comp)

\[ A[N] = f(A'[N - 1], I[N]) \]

(Gen)

\[ i = N \]

\[ A[i] = f(A'[N - 1], I[N]) \]

\[ i = 0 \]

(Comp)

\[ I[0] = f(A'[N - 1], I[N]) \]

\[ i > 0 \]

success if \( i = N \)

\[ f(A[i - 1], I[i]) = f(A'[N - 1], I[N]) \]

(Dec)

\[ I[i] = I[N] \]

\[ A[i - 1] = A'[N - 1] \]

(Gen)

\[ i = i - 1 \]

\[ i = i - 1 \]

\[ i' = i' - 1 \]

\[ A[i] = A'[i'] \]

\[ i = 0, i' = 0 \]

\[ I[0] = I[0] \]

success

\[ i > 0, i' > 0 \]

\[ f(A[i - 1], I[i]) = f(A'[i' - 1], I[i']) \]

\[ i = i' \]

success if \( i = i' \)

\[ I[i] = I[i'] \]

\[ O = A[N] \]

\[ A[i] = \begin{cases} 
  i > 0 & : f(A[i - 1], I[i]) \\
  i = 0 & : I[0] 
\end{cases} \]

\[ A'[i] = \begin{cases} 
  i > 0 & : f(A'[i - 1], I[i]) \\
  i = 0 & : I[0] 
\end{cases} \]

Precondition: \( N > 0 \)

\[ O = f(A'[N - 1], I[N]) \]

\[ A'[i] = \begin{cases} 
  i > 0 & : f(A'[i - 1], I[i]) \\
  i = 0 & : I[0] 
\end{cases} \]

- **Failure states** are unreachable
- **Success states** \( I[i] = I[i'] \) are reached with \( i = i' \)
Semantic factorizations

<table>
<thead>
<tr>
<th>Expression factorization</th>
<th>Constraint factorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u \xrightarrow{\Delta} v$</td>
<td>$u \xrightarrow{\Delta} (v &lt; 0)$</td>
</tr>
</tbody>
</table>

Cost model

For a DAG $\mathcal{D} = (N, E)$: $|\mathcal{D}| = \sum_{n \in N} w(n) \cdot bw(n)$
**Kernels:** PolyBench/C v3.2

**Target:** FPGA Arria 10 10AX115S2F4I1SG

**Intel Quartus Prime TM 16.1.2**

**Optimization per process**

**Compare:**

- **SEM+Quartus:** Our DAG, pipelined
- **Quartus:** Direct implementation (affine control in VHDL) + output registers

<table>
<thead>
<tr>
<th>kernel</th>
<th>#dags</th>
<th>#C</th>
<th>#E</th>
</tr>
</thead>
<tbody>
<tr>
<td>2mm</td>
<td>15</td>
<td>250</td>
<td>161</td>
</tr>
<tr>
<td>3mm</td>
<td>18</td>
<td>369</td>
<td>206</td>
</tr>
<tr>
<td>atax</td>
<td>12</td>
<td>134</td>
<td>83</td>
</tr>
<tr>
<td>bicg</td>
<td>11</td>
<td>112</td>
<td>69</td>
</tr>
<tr>
<td>correlation</td>
<td>27</td>
<td>356</td>
<td>205</td>
</tr>
<tr>
<td>covariance</td>
<td>16</td>
<td>243</td>
<td>143</td>
</tr>
<tr>
<td>doitgen</td>
<td>9</td>
<td>145</td>
<td>124</td>
</tr>
<tr>
<td>fdtd-2d</td>
<td>13</td>
<td>502</td>
<td>167</td>
</tr>
<tr>
<td>gemm</td>
<td>10</td>
<td>125</td>
<td>93</td>
</tr>
<tr>
<td>gemver</td>
<td>20</td>
<td>187</td>
<td>137</td>
</tr>
<tr>
<td>gesummv</td>
<td>14</td>
<td>95</td>
<td>84</td>
</tr>
<tr>
<td>heat-3d</td>
<td>8</td>
<td>734</td>
<td>175</td>
</tr>
<tr>
<td>jacob-1d</td>
<td>8</td>
<td>134</td>
<td>64</td>
</tr>
<tr>
<td>jacob-2d</td>
<td>8</td>
<td>370</td>
<td>111</td>
</tr>
<tr>
<td>lu</td>
<td>7</td>
<td>213</td>
<td>87</td>
</tr>
<tr>
<td>mvt</td>
<td>11</td>
<td>118</td>
<td>70</td>
</tr>
<tr>
<td>seidel-2d</td>
<td>5</td>
<td>226</td>
<td>63</td>
</tr>
<tr>
<td>symm</td>
<td>13</td>
<td>213</td>
<td>116</td>
</tr>
<tr>
<td>syr2k</td>
<td>10</td>
<td>135</td>
<td>90</td>
</tr>
<tr>
<td>syrk</td>
<td>9</td>
<td>118</td>
<td>81</td>
</tr>
<tr>
<td>trisolv</td>
<td>9</td>
<td>93</td>
<td>56</td>
</tr>
<tr>
<td>trmm</td>
<td>8</td>
<td>118</td>
<td>79</td>
</tr>
</tbody>
</table>