CHAIR FOR COMPILER CONSTRUCTION

Faster MPSoC Task Mapping via Symmetry Reduction

Timo Nicolai
timo.nicolai@mailbox.tu-dresden.de

June 9, 2020

Contents

1. Inspirations

Contents

1. Inspirations
2. Problem Statement

Contents

1. Inspirations
2. Problem Statement
3. Symmetry Reduction

Contents

1. Inspirations
2. Problem Statement
3. Symmetry Reduction
4. Representing and Extracting Symmetries

Contents

1. Inspirations
2. Problem Statement
3. Symmetry Reduction
4. Representing and Extracting Symmetries
5. The TMOR Problem

Contents

1. Inspirations
2. Problem Statement
3. Symmetry Reduction
4. Representing and Extracting Symmetries
5. The TMOR Problem
6. Experimental Results

Contents

1. Inspirations
2. Problem Statement
3. Symmetry Reduction
4. Representing and Extracting Symmetries
5. The TMOR Problem
6. Experimental Results
7. Further Research

Inspirations

- Original idea: [Goens et al., 2017]

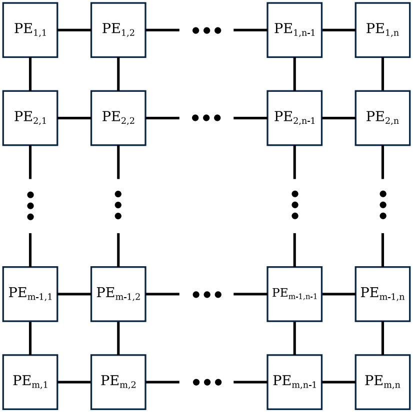
Inspirations

- Original idea: [Goens et al., 2017]
- Theoretical foundations: [Holt, 2005] and [East et al., 2019]

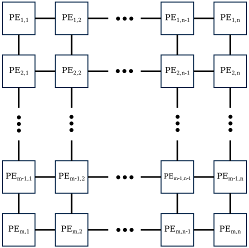
Inspirations

- Original idea: [Goens et al., 2017]
- Theoretical foundations: [Holt, 2005] and [East et al., 2019]
- Important optimizations: [Donaldson and Miller, 2009]

Problem Statement

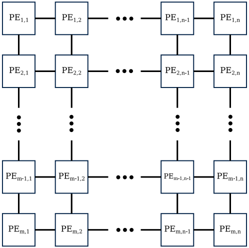


Problem Statement



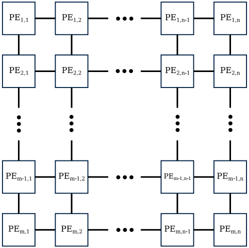
- Want to intelligently map tasks to processing elements

Problem Statement



- Want to intelligently map tasks to processing elements
- Best choice depends on underlying optimality criteria

Problem Statement



- Want to intelligently map tasks to processing elements
- Best choice depends on underlying optimality criteria
- Need to perform costly simulation!

Symmetry Reduction

- One approach:
  - Generate promising mappings based on previous simulations
  - $\rightarrow$ Traverse search space "intelligently"

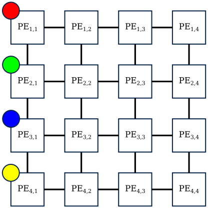
Symmetry Reduction

- One approach:
  - Generate promising mappings based on previous simulations
  - $\rightarrow$ Traverse search space "intelligently"
- Another approach:
  - Partition search space *by (partial) symmetry*
  - Only work with representative of each partition
  - $\rightarrow$ "Collapse" search space

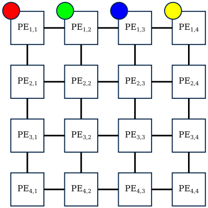
Symmetry Reduction

- One approach:
  - Generate promising mappings based on previous simulations
  - $\rightarrow$ Traverse search space "intelligently"
- Another approach:
  - Partition search space *by (partial) symmetry*
  - Only work with representative of each partition
  - $\rightarrow$ "Collapse" search space
- Both approaches can be combined!

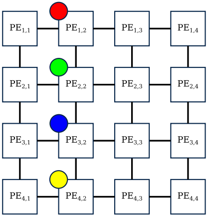
Symmetry Reduction



(a)

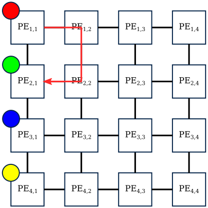


(b)

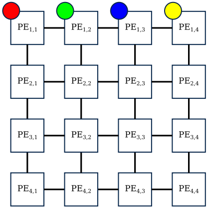


(c)

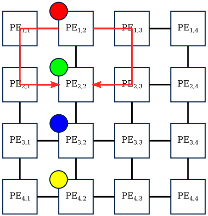
Symmetry Reduction



(a)

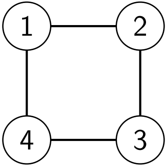


(b)

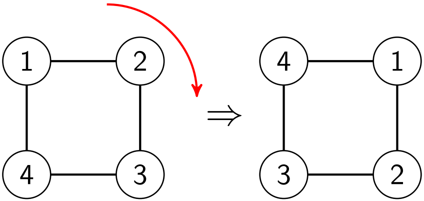


(c)

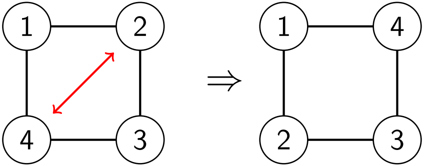
Representing Symmetries: Automorphism Groups



Representing Symmetries: Automorphism Groups

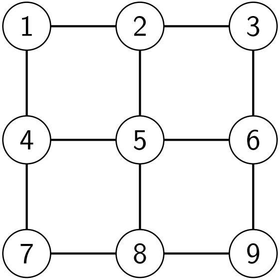


$\Rightarrow (1\ 2\ 3\ 4)$



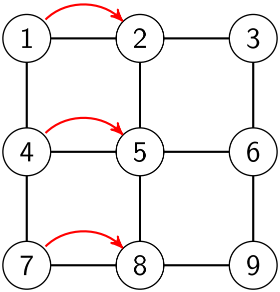
$\Rightarrow (2\ 4)$

Representing Partial Symmetries: Partial Automorphism Inverse Monoids



CHAIR FOR COMPILER CONSTRUCTION

Representing Partial Symmetries: Partial Automorphism Inverse Monoids



$\Rightarrow$ [1 2][4 5][7 8]

Extracting (Partial) Symmetries

- Describe MPSoC architecture as *architecture graph*

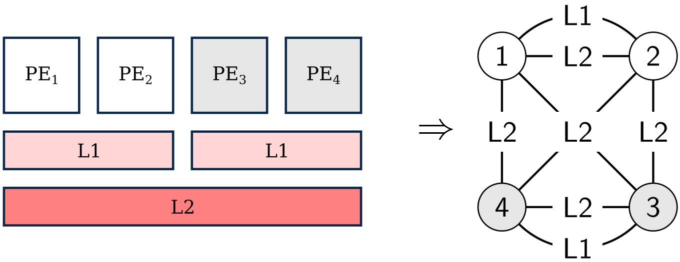
Extracting (Partial) Symmetries

- Describe MPSoC architecture as *architecture graph*
- Determine Automorphism Group G:
  - Transform into "equivalent" vertex colored graph
  - Use *nauty* [McKay and Piperno, 2014]

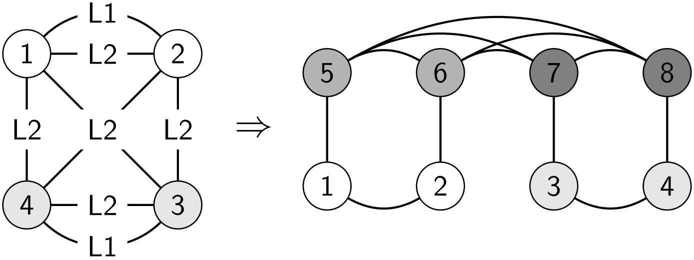
Extracting (Partial) Symmetries

- Describe MPSoC architecture as *architecture graph*
- Determine Automorphism Group G:
  - Transform into “equivalent” vertex colored graph
  - Use *nauty* [McKay and Piperno, 2014]
- Determine Partial Automorphism Inverse Monoid M:
  - Construct search tree of possible generators
  - Prune certain subtrees
  - Not efficient enough in practice

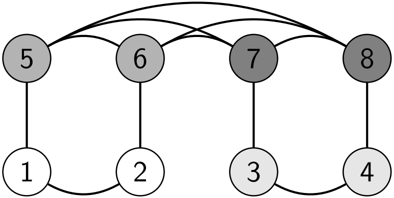
Extracting (Partial) Symmetries



Extracting (Partial) Symmetries



Extracting (Partial) Symmetries



$$\Rightarrow \{(1\ 2)\cancel{(5\ 6)}, (3\ 4)\cancel{(7\ 8)}\}$$

Extracting (Partial) Symmetries

$$\{(1\ 2), (3\ 4)\} \Rightarrow \begin{array}{l} \text{Base: } [1, 3] \\ \text{Strong Generating Set: } \{(1\ 2), (3\ 4)\} \end{array}$$

The TMOR Problem I: Mapping Orbits

- Represent mappings by k-tuples $t = (t_1, t_2, \ldots, t_k)$

The TMOR Problem I: Mapping Orbits

- Represent mappings by k-tuples $t = (t_1, t_2, \ldots, t_k)$
- Define *orbit* $t^G = \{((g(t_1), g(t_2), \ldots, g(t_k)) \mid g \in G\}$

The TMOR Problem I: Mapping Orbits

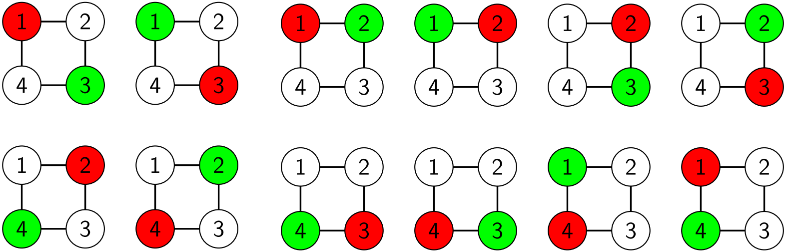
- Represent mappings by k-tuples $t = (t_1, t_2, \ldots, t_k)$
- Define *orbit* $t^G = \{((g(t_1), g(t_2), \ldots, g(t_k)) \mid g \in G\}$
- Orbits **partition** search space

The TMOR Problem I: Mapping Orbits

- Represent mappings by k-tuples $t = (t_1, t_2, \ldots, t_k)$
- Define *orbit* $t^G = \{((g(t_1), g(t_2), \ldots, g(t_k)) \mid g \in G\}$
- Orbits **partition** search space
- Reduce search space to set of *canonical orbit representatives*

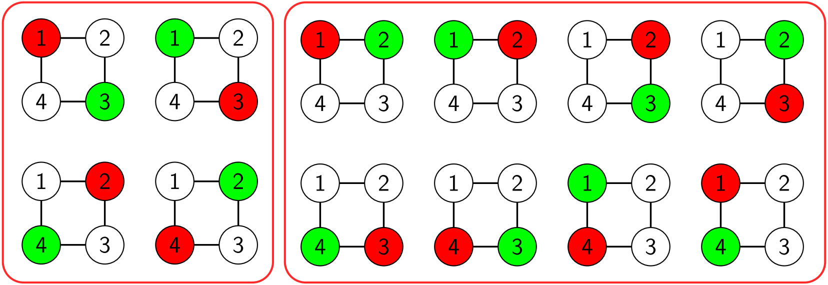
The TMOR Problem I: Mapping Orbits

$$G = \langle\{(1\ 2\ 3\ 4), (2\ 4)\}\rangle$$



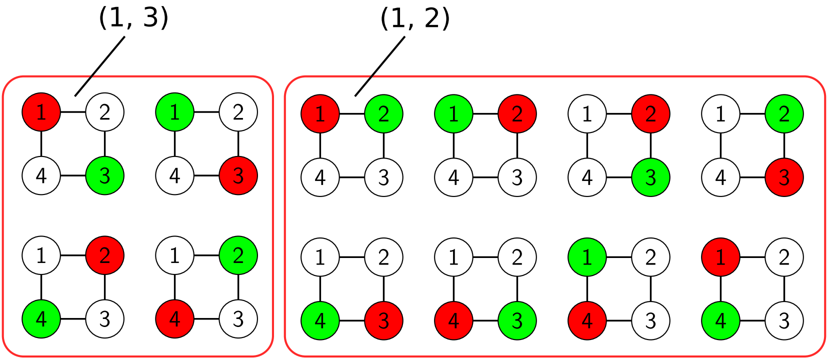
The TMOR Problem I: Mapping Orbits

$$G = \langle\{(1\ 2\ 3\ 4), (2\ 4)\}\rangle$$



The TMOR Problem I: Mapping Orbits

$$G = \langle \{(1\ 2\ 3\ 4), (2\ 4)\} \rangle$$



CHAIR FOR COMPILER CONSTRUCTION

The TMOR Problem I: Mapping Orbits

- Determining all orbits usually too costly

The TMOR Problem I: Mapping Orbits

- Determining all orbits usually too costly
- Iteratively determine and hash canonical representatives

The TMOR Problem I: Mapping Orbits

- Determining all orbits usually too costly
- Iteratively determine and hash canonical representatives

procedure ORBIT_IDENTIFIER(t, reprs)

  repr $\leftarrow$ TMOR(t)

  if repr $\notin$ reprs **then**

    Append repr to reprs

  end if

  return index of repr in reprs

end procedure

The TMOR Problem II: Finding Canonical Orbit Representatives

- First approach: explicit orbit enumeration

```
procedure TMOR_ORBIT(t, G = ⟨S⟩)
    orbit ← {}

    while orbit is changing do
        for t′ ∈ orbit, g in S do
            orbit ← orbit ∪ {g(t′)}
        end for
    end while

    return min(orbit)
end procedure
```

The TMOR Problem II: Finding Canonical Orbit Representatives

- Second approach: group enumeration

```
procedure TMOR_ITERATE(t, G)
    repr ← t

    for g ∈ G do
        if g(t) < repr then
            repr ← g(t)
        end if
    end for

    return repr
end procedure
```

The TMOR Problem II: Finding Canonical Orbit Representatives

- Third approach: local search

```
procedure TMOR_LOCAL_SEARCH(t, G = ⟨S⟩)
    repr ← t

    while repr is changing do
        repr ← min({g(repr) | g ∈ S})
    end while

    return repr
end procedure
```

The TMOR Problem III: Optimization by Decomposition

- Many architecture graphs are *separable* or *hierarchical*

The TMOR Problem III: Optimization by Decomposition

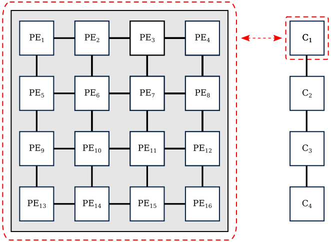
- Many architecture graphs are *separable* or *hierarchical*
- Separable: any heterogeneous archictecture

The TMOR Problem III: Optimization by Decomposition

- Many architecture graphs are *separable* or *hierarchical*
- Separable: any heterogeneous archictecture
- Hierarchical: HAEC

The TMOR Problem III: Optimization by Decomposition

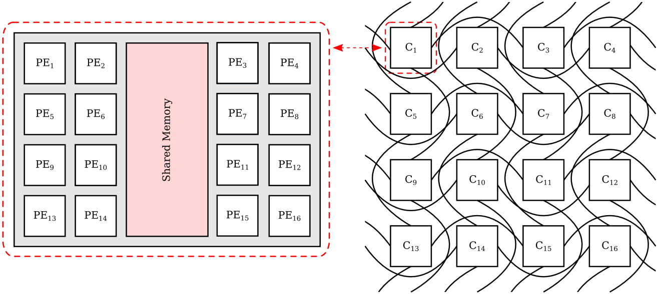
- Many architecture graphs are *separable* or *hierarchical*
- Separable: any heterogeneous archictecture
- Hierarchical: **HAEC**



CHAIR FOR COMPILER CONSTRUCTION

The TMOR Problem III: Optimization by Decomposition

- Many architecture graphs are *separable* or *hierarchical*
- Separable: any heterogeneous archictecture
- Hierarchical: HAEC, **Kalray**



The TMOR Problem III: Optimization by Decomposition

- Automorphism groups of such graphs decompose into:

The TMOR Problem III: Optimization by Decomposition

- Automorphism groups of such graphs decompose into:
  - Separable architectures: *direct products*
    $G = G_1 \times G_2$, $|G| = |G_1| \cdot |G_2| \cdots |G_n|$
  - Hierarchical architectures: *wreath products*
    $G = G_{\text{proto}} \wr G_{\text{super}}$, $|G| = |G_{\text{proto}}|^{\deg(G_{\text{super}})} \cdot |G_{\text{super}}|$

The TMOR Problem III: Optimization by Decomposition

- Automorphism groups of such graphs decompose into:
  - Separable architectures: *direct products*
    $G = G_1 \times G_2$, $|G| = |G_1| \cdot |G_2| \cdots |G_n|$
  - Hierarchical architectures: *wreath products*
    $G = G_{\mathrm{proto}} \wr G_{\mathrm{super}}$, $|G| = |G_{\mathrm{proto}}|^{\deg(G_{\mathrm{super}})} \cdot |G_{\mathrm{super}}|$
- Solve TMOR problem separately for components

The TMOR Problem III: Optimization by Decomposition

- Automorphism groups of such graphs decompose into:
  - Separable architectures: *direct products*
    $G = G_1 \times G_2,\ |G| = |G_1| \cdot |G_2| \cdots |G_n|$
  - Hierarchical architectures: *wreath products*
    $G = G_{\mathrm{proto}} \wr G_{\mathrm{super}},\ |G| = |G_{\mathrm{proto}}|^{\deg(G_{\mathrm{super}})} \cdot |G_{\mathrm{super}}|$
- Solve TMOR problem separately for components
- Decomposition specified by user or detected automatically

The TMOR Problem III: Optimization by Decomposition

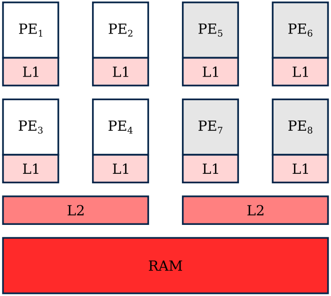
- Automorphism groups of such graphs decompose into:
  - Separable architectures: *direct products*
    $G = G_1 \times G_2$, $|G| = |G_1| \cdot |G_2| \cdots |G_n|$
  - Hierarchical architectures: *wreath products*
    $G = G_{\mathrm{proto}} \wr G_{\mathrm{super}}$, $|G| = |G_{\mathrm{proto}}|^{\mathrm{deg}(G_{\mathrm{super}})} \cdot |G_{\mathrm{super}}|$
- Solve TMOR problem separately for components
- Decomposition specified by user or detected automatically
- Idea based on [Donaldson and Miller, 2009]

Experimental Results I: Architectures

- Experiments run for: **Exynos**

Experimental Results I: Architectures

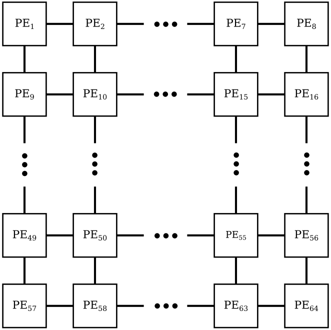
- Experiments run for: Exynos , **Parallella**



CHAIR FOR COMPILER CONSTRUCTION

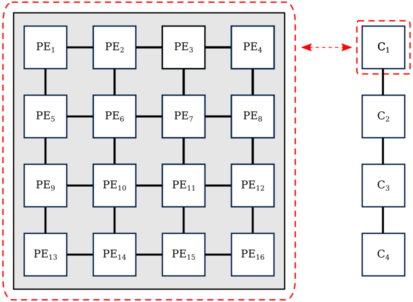
Experimental Results I: Architectures

- Experiments run for: Exynos , Parallella , **HAEC**



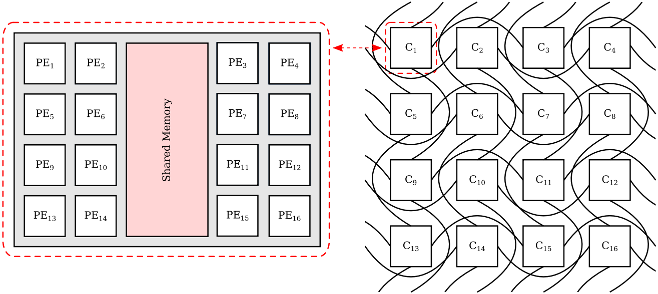
Experimental Results I: Architectures

- Experiments run for: Exynos , Parallella , HAEC , **Kalray**

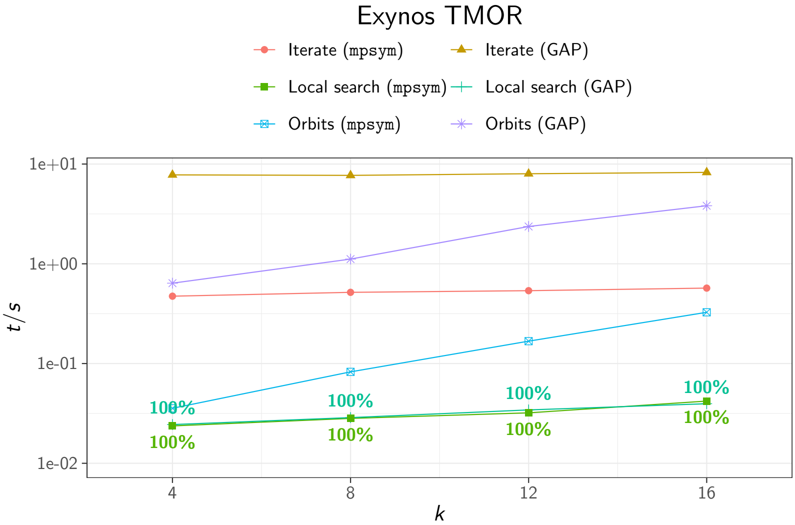


Experimental Results I: Architectures

- Experiments run for: Exynos , Parallella , HAEC , Kalray

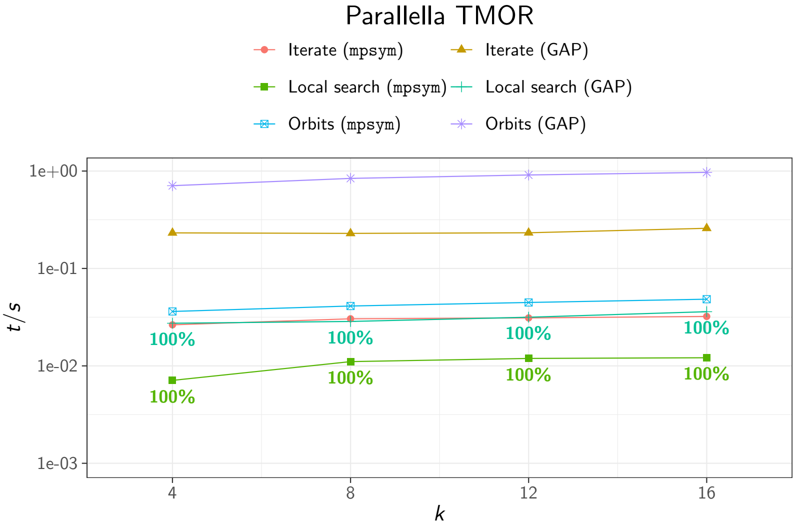


Experimental Results II: TMOR



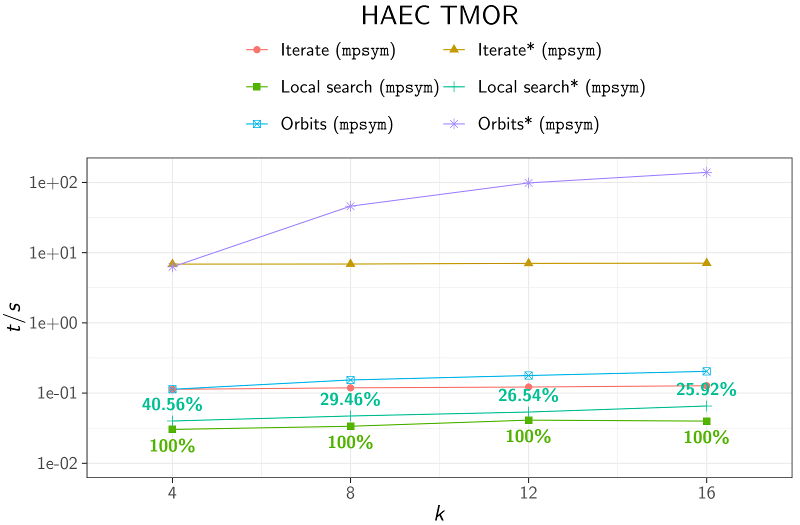
CHAIR FOR COMPILER CONSTRUCTION

Experimental Results II: TMOR

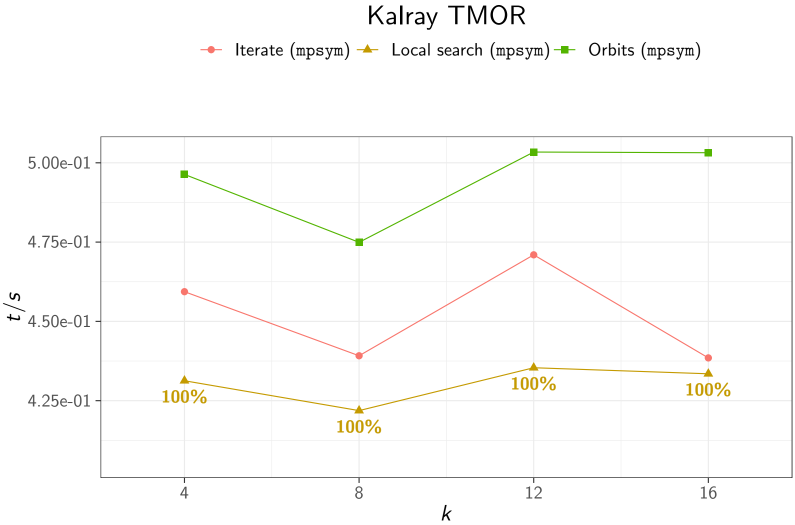


CHAIR FOR COMPILER CONSTRUCTION

Experimental Results II: TMOR



Experimental Results II: TMOR



Experimental Results II: TMOR

- Lessons learned:
  - Performance of orbit enumeration depends strongly on k

Experimental Results II: TMOR

- Lessons learned:
  - Performance of orbit enumeration depends strongly on k
  - Local search can be fast and accurate

Experimental Results II: TMOR

- Lessons learned:
  - Performance of orbit enumeration depends strongly on k
  - Local search can be fast and accurate
  - Decomposition can be very powerful

Experimental Results II: TMOR

- Lessons learned:
  - Performance of orbit enumeration depends strongly on k
  - Local search can be fast and accurate
  - Decomposition can be very powerful
  - `mpsym` outperforms GAP

Further Research

- Heuristic local search

Further Research

- Heuristic local search
- Partial Automorphism discovery

Further Research

- Heuristic local search
- Partial Automorphism discovery
- Inverse Monoid enumeration

Further Research

- Heuristic local search
- Partial Automorphism discovery
- Inverse Monoid enumeration
- Interfacing GAP and C++

The End

Thank you for your attention!

References

Donaldson, A. F. and Miller, A. (2009).
On the constructive orbit problem.
Ann Math Atrif Intell, 57:1–35.

East, J., Egri-Nagy, A., Mitchell, J., and Péresse, Y. (2019).
Computing finite semigroups.
Journal of Symbolic Computation, 92:110–155.

Goens, A., Siccha, S., and Castrillon, J. (2017).
Symmetry in software synthesis.
ACM Trans. Archit. Code Optim., 14(2).

Holt, D. F. (2005).
Handbook of Computational Group Theory.
CRC Press.

McKay, B. D. and Piperno, A. (2014).
Practical graph isomorphism, ii.
Journal of Symbolic Computation, 60:94 – 112.

Nicolai, T. (2020).
mpsym.
`https://github.com/Time0o/mpsym`.

CHAIR FOR COMPILER CONSTRUCTION