Clustering and Alignment Methods for Structural Comparison of Parallel Applications

Scalable Tools Workshop 2016
Lake Tahoe, California, USA

Matthias Weber

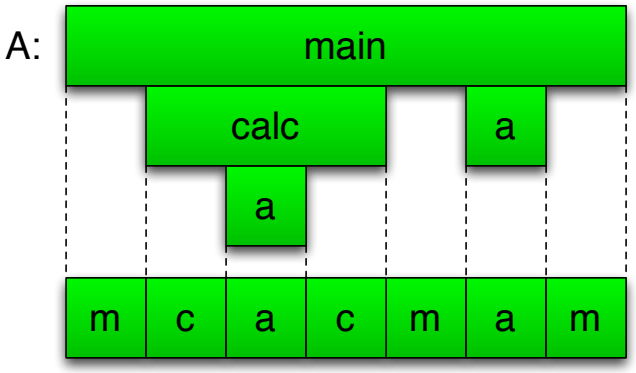
August 1, 2016

Outline

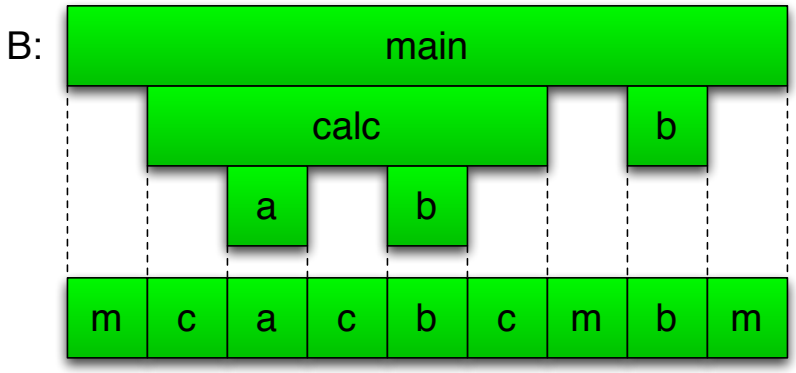
- Motivation
- Structural Clustering of Processes
  - Determining Similarity
  - Efficient Computation and Storage of Clusters
  - Applicability Study
- Structural Alignment of Processes
  - Alignment of Multiple Process Sequences
  - Merged Call Graph
- Conclusion

Manual comparison of two processes: Default vs. optimized application run

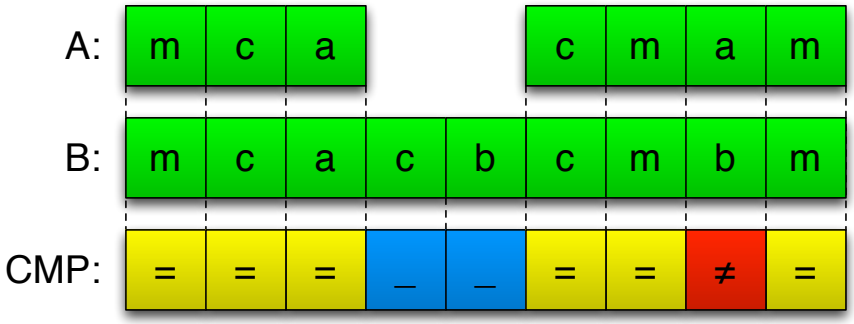
- Manual comparison of process event streams is extremely challenging due to the large number of events and the need to correctly line up trace events
- Automatic support for event-wise trace comparison needed



Construction of sequence A



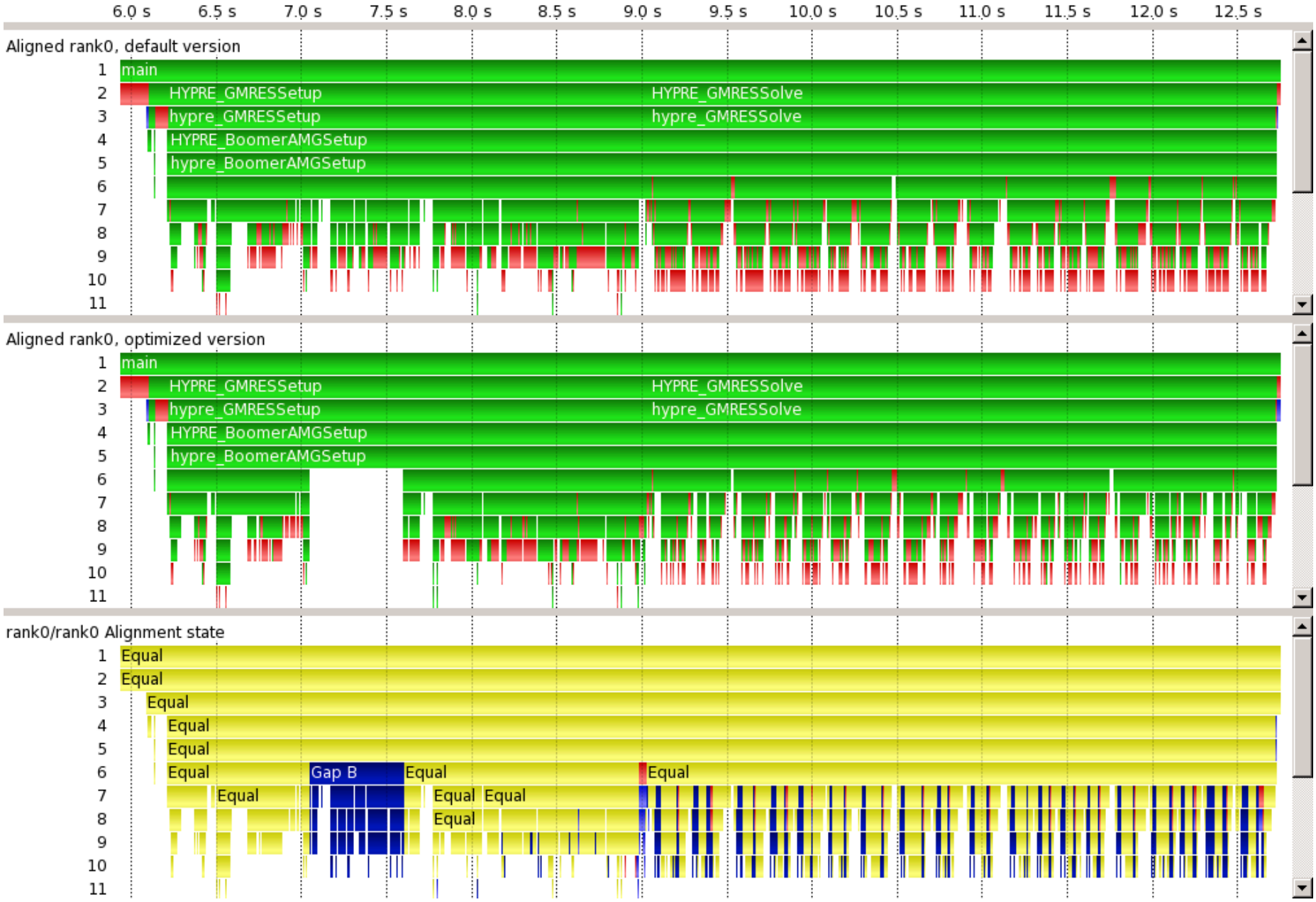
Construction of sequence B

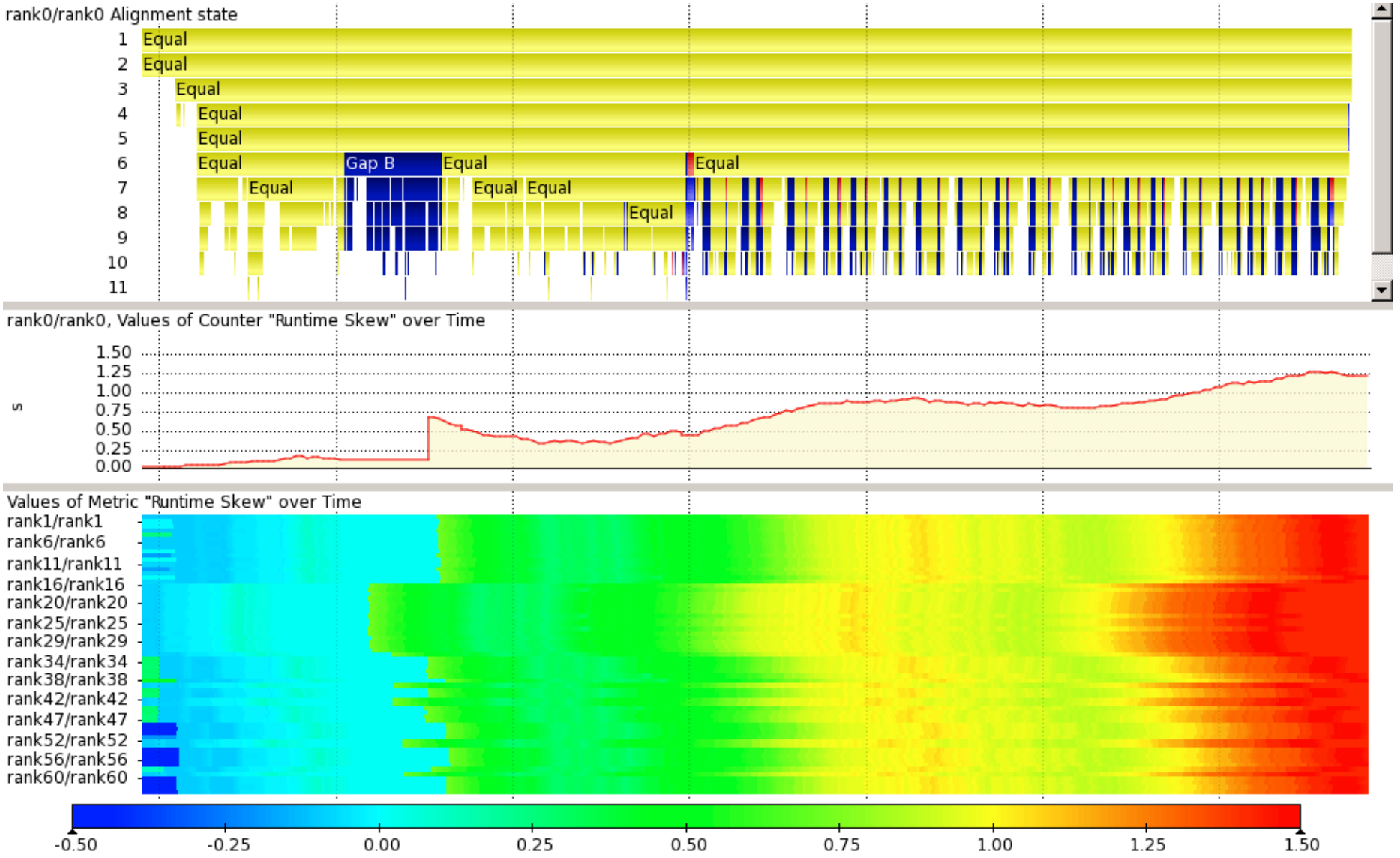


Resulting alignment of sequences A and B [1]

Pairwise Structural Comparisons with Sequence Alignment Methods

- Sequence alignment allows to compare process structure in detail
- Pairwise comparisons expose differences between two processes
- The pairwise process comparison is computationally expensive, forbidding exhaustive comparison of all process combinations


6.0 s
6.5 s
7.0 s
7.5 s
8.0 s
8.5 s
9.0 s
9.5 s
10.0 s
10.5 s
11.0 s
11.5 s
12.0 s
12.5 s
Aligned rank0, default version
1 main
2 HYPRE_GMRESSetup
HYPRE_GMRESSolve
3 hypre_GMRESSetup
hypre_GMRESSolve
4 HYPRE_BoomerAMGSetup
5 hypre_BoomerAMGSetup
6
7
8
9
10
11
Aligned rank0, optimized version
1 main
2 HYPRE_GMRESSetup
HYPRE_GMRESSolve
3 hypre_GMRESSetup
hypre_GMRESSolve
4 HYPRE_BoomerAMGSetup
5 hypre_BoomerAMGSetup
6
7
8
9
10
11
rank0/rank0 Alignment state
1 Equal
2 Equal
3 Equal
4 Equal
5 Equal
6 Equal
Gap B
Equal
Equal
7 Equal
Equal
Equal
8 Equal
9
10
11



[2]

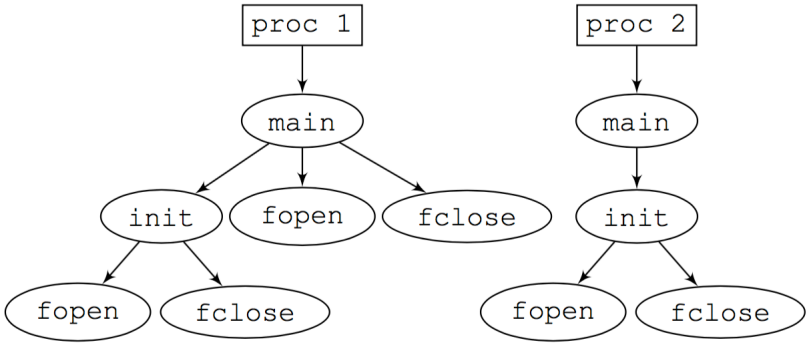
AMG2006: Runtime analysis

- The optimized version runs faster and finishes about 1.25 seconds earlier

Outline

- Motivation
- **Structural Clustering of Processes**
  - Determining Similarity
  - Efficient Computation and Storage of Clusters
  - Applicability Study
- Structural Alignment of Processes
  - Alignment of Multiple Process Sequences
  - Merged Call Graph
- Conclusion

Call trees for two example processes:



Function pairs of proc 1:

$$pairs(\texttt{proc 1}) = \{\epsilon \rightarrow \texttt{main}, \texttt{main} \rightarrow \texttt{init}, \texttt{main} \rightarrow \texttt{fopen}, \texttt{main} \rightarrow \texttt{fclose}, \texttt{init} \rightarrow \texttt{fopen}, \texttt{init} \rightarrow \texttt{fclose}\}$$

Function pairs of proc 2:

$$pairs(\texttt{proc 2}) = \{\epsilon \rightarrow \texttt{main}, \texttt{main} \rightarrow \texttt{init}, \texttt{init} \rightarrow \texttt{fopen}, \texttt{init} \rightarrow \texttt{fclose}\}$$

Definition of structural similarity based on function pairs:

$$pairsim(P_1, P_2) := \frac{|\, pairs(P_1) \cap pairs(P_2) \,|}{|\, pairs(P_1) \cup pairs(P_2) \,|}$$

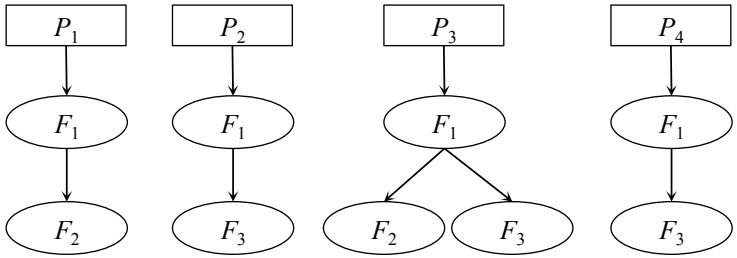
Function pair similarity of the two example processes:

$$pairsim(\texttt{proc 1}, \texttt{proc 2}) = \frac{4}{6}$$

Structural Similarity Measure

- Structural information is contained in call trees (disregarding timing)
  - Easily obtainable from call-path profiles or traces
- Differences between processes are based function pairs that represent the *caller-callee* relation:
  - *pairs*(*P*) := {(F_1, F_2) : F_1 calls F_2 on *P* at least once}
- Measure is independent of: number of calls, number of iterations, recursion depth, timing
- Assumption: static executable → increasing process count or problem size does not increase the number of function pairs

Four example processes:



Formal context:

$(\mathcal{P}, \mathcal{A} \subseteq \mathcal{F} \times \mathcal{F}, \mathcal{I})$

With:

$$\mathcal{P} := \{P_1, P_2, P_3, P_4\},$$
$$\mathcal{F} := \{F_1, F_2, F_3\},$$
$$\mathcal{A} := \{\epsilon \to F_1, F_1 \to F_2, F_1 \to F_3\},$$
$$\mathcal{I} := \{(P_1, \epsilon \to F_1),\ (P_2, F_1 \to F_2),\ (P_2, \epsilon \to F_1),\ (P_2, F_1 \to F_3),\ (P_3, \epsilon \to F_1),\ (P_3, F_1 \to F_2),\ (P_3, F_1 \to F_3),\ (P_4, \epsilon \to F_1),\ (P_4, F_1 \to F_3)\ \}$$

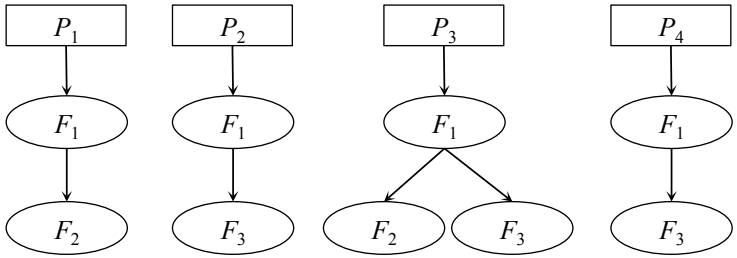
Incidence relation table for the example processes: [3]

	$\epsilon \to F_1$	$F_1 \to F_2$	$F_1 \to F_3$
P_1	×	×	
P_2	×		×
P_3	×	×	×
P_4	×		×

Formal Context

- The function pair similarity measure is set-based and allows to use *formal concept analysis* methods
- Similarity data can be described as a *formal context* [4], which is a triple (*O,A,I*), where
  - *O* is a set of objects,
  - *A* a set of attributes,
  - and $I \subseteq O \times A$ an incidence relation associating objects with attributes
- Storing the information of all function pairs in a table is not scalable

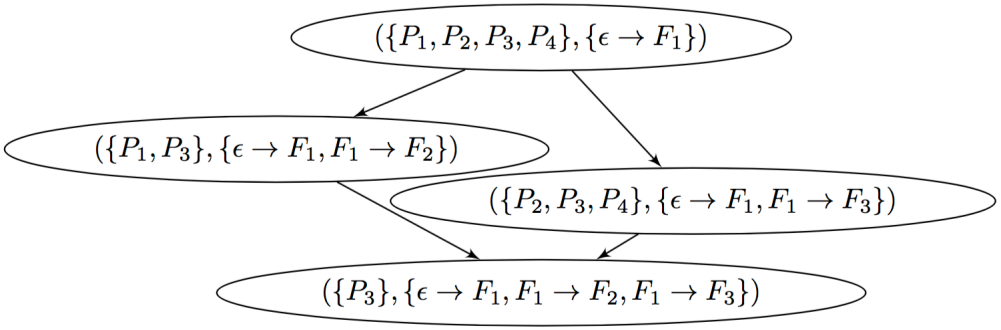
Four example processes:



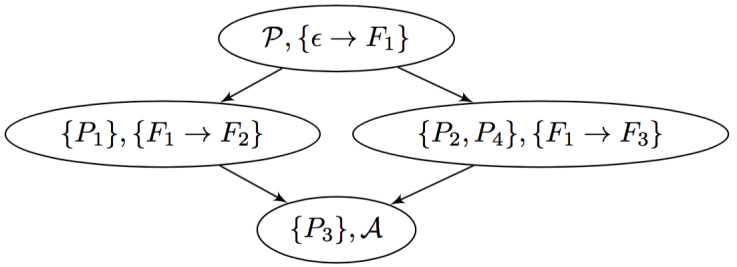
Incidence relation table for the example processes:

	$\epsilon \to F_1$	$F_1 \to F_2$	$F_1 \to F_3$
P_1	×	×	
P_2	×		×
P_3	×	×	×
P_4	×		×

Concept lattice with redundant information:



Concept lattice without redundant information:

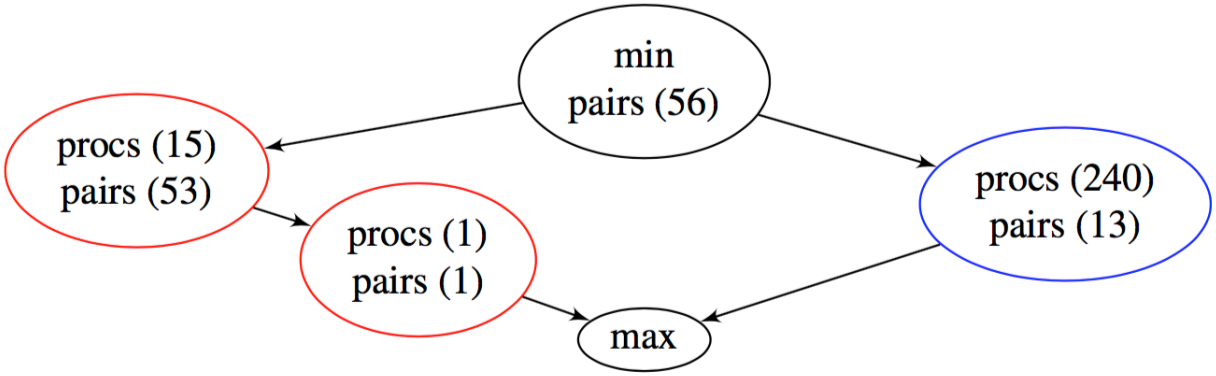


[3]

Concept Lattice

- Concept lattices order and store formal contexts efficiently
- Similar processes are grouped during construction
- Lattices have a small memory footprint; each process and each function pair occurs exactly once in the lattice
- Lattice construction is done using the algorithm from van der Merwe [5], that allows iterative adding of processes
- Expected complexity for building and storing the lattice is linear, the worst case is complexity is exponential

Concept lattice for BT with 16 MPI processes (red) and 15 OpenMP threads (blue) per process:



[3]

Concept Lattice for BT-MZ

- 256 processes in total
- 3 groups
  - Two groups with MPI processes (red)
  - One group with OpenMP threads (blue)
- All processes share 56 function pairs
- No process executes all function pairs

Application			Result		
Name	Processes	Func. pairs	t_{eval} (ms)	Lattice Nodes	Process Groups
HPL	2,360	8	< 10	2	2
GROMACS	36	1,381	< 10	24	11
CCLM	180	180	< 10	6	3
COSMO-SPECS	100	50	< 10	1	1
WRF	64	774	< 10	2	2
FD4	65,536	55	55	22	14
HOMME	1,024	179	< 10	3	3
AMG2006	1,024	440	66	11	7
IRS	64	989	34	18	7
LULESH	432	406	49	182	35
ParaDiS	128	649	3,486	6,367	74
PIConGPU	39	474	11	60	17
BT-MZ	16	126	< 10	5	3
HPCC MPI-FFTE	128	109	70	7	4
PEPC	16,384	113	15	2	2

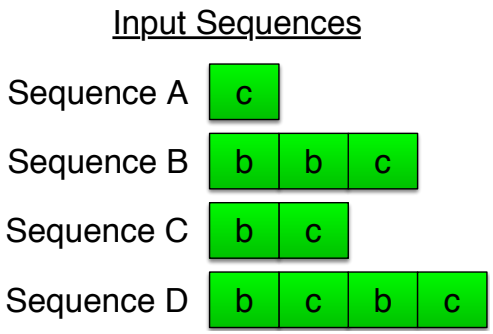
[3]

Applicability Study

- Study using 15 HPC applications with different characteristics
- t_{eval} denotes the time to construct the lattice containing all processes and to compute the similarity matrix
- For all applications except ParaDiS t_{eval} is below 0.1 seconds
- For 10 applications the number of process groups is below 10

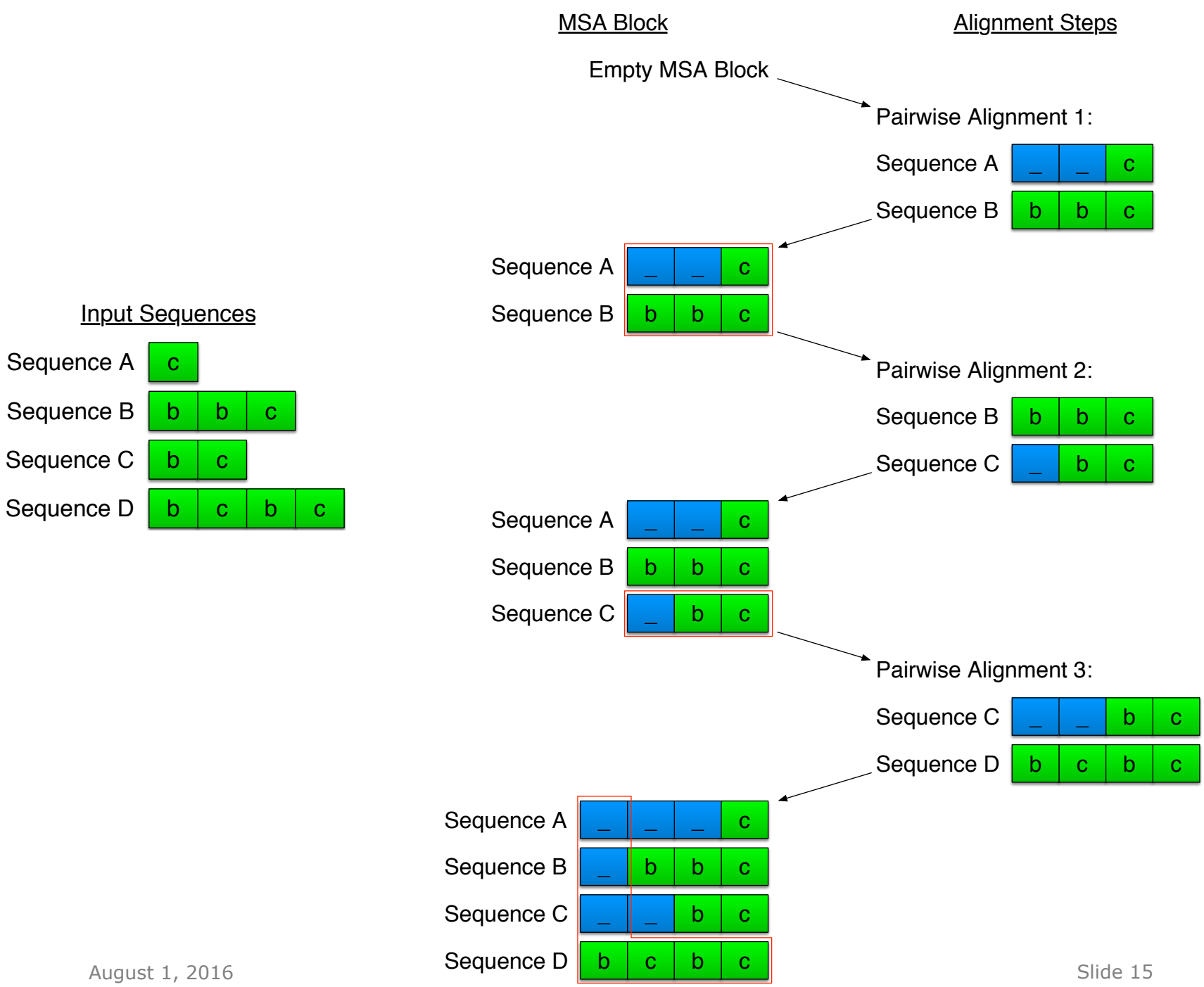
Outline

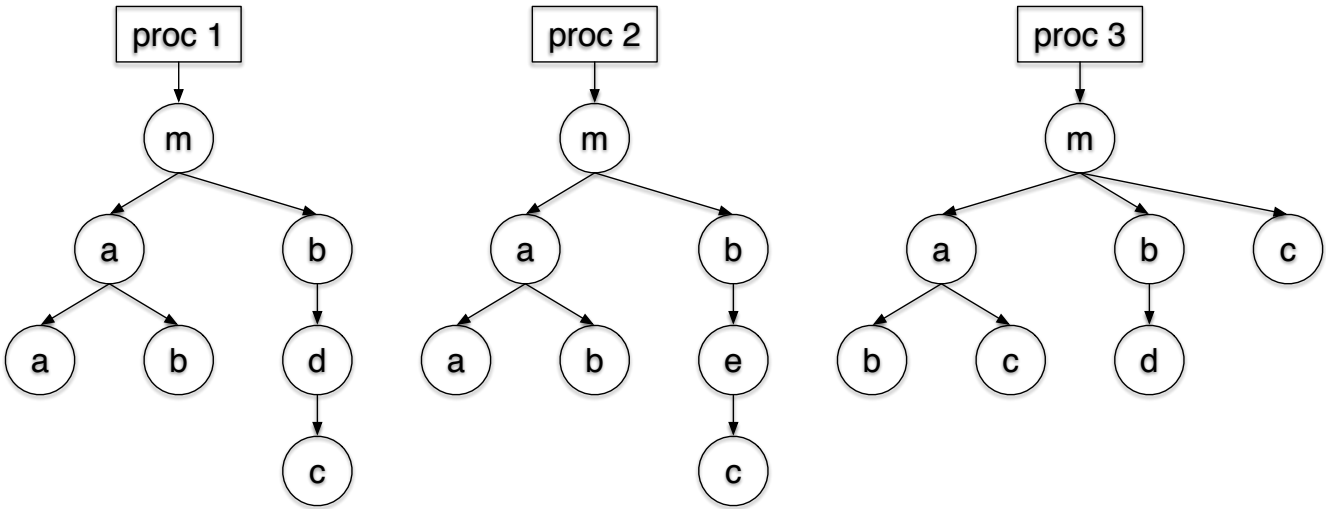
- Motivation
- Structural Clustering of Processes
  - Determining Similarity
  - Efficient Computation and Storage of Clusters
  - Applicability Study
- **Structural Alignment of Processes**
  - Alignment of Multiple Process Sequences
  - Merged Call Graph
- Conclusion



Structural Comparisons of Multiple Sequences

- Progressive multiple sequence alignment (MSA) can align many sequences to one alignment block
- Progressively applied pairwise alignments add new sequences to the alignment block
- Structural pre-clustering helps to select processes for comparison
- MSA allows to compare all processes of a cluster in detail

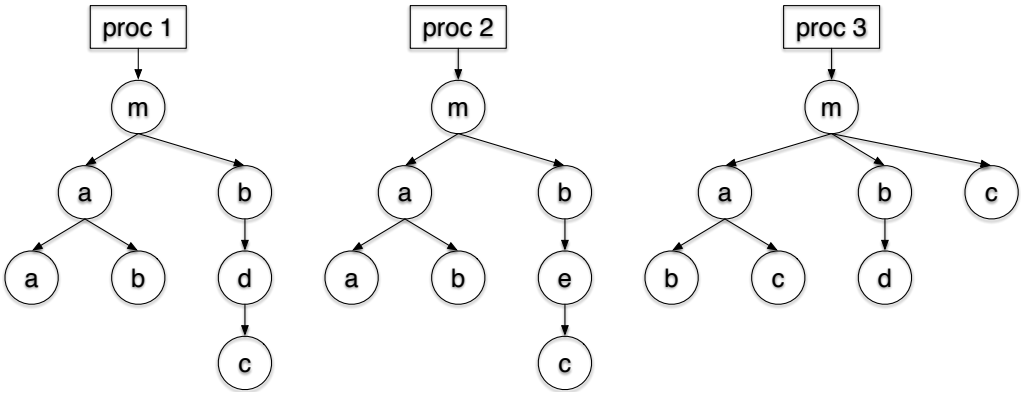

MSA Block
Alignment Steps
Empty MSA Block
Pairwise Alignment 1:
Sequence A _ _ c
Sequence B b b c
Sequence A _ _ c
Sequence B b b c
Pairwise Alignment 2:
Sequence B b b c
Sequence C _ b c
Sequence A _ _ c
Sequence B b b c
Sequence C _ b c
Pairwise Alignment 3:
Sequence C _ _ b c
Sequence D b c b c
Sequence A _ _ _ c
Sequence B _ b b c
Sequence C _ _ b c
Sequence D b c b c
Input Sequences
Sequence A c
Sequence B b b c
Sequence C b c
Sequence D b c b c



Three input processes

Hierarchical Multiple Sequence Alignment Approach

- Aligning full process sequences is too computationally expensive
- The hierarchical approach exploits the call-tree structure, and splits up process sequences into several smaller sub-sequences



Step

Merged Process-Tree

Multiple Sequence Alignments

1

proc 1/2/3

proc 1 m

proc 2 m

proc 3 m

2

proc 1/2/3

m

proc 1 a b _

proc 2 a b _

proc 3 a b c

3

proc 1/2/3

m

a b c

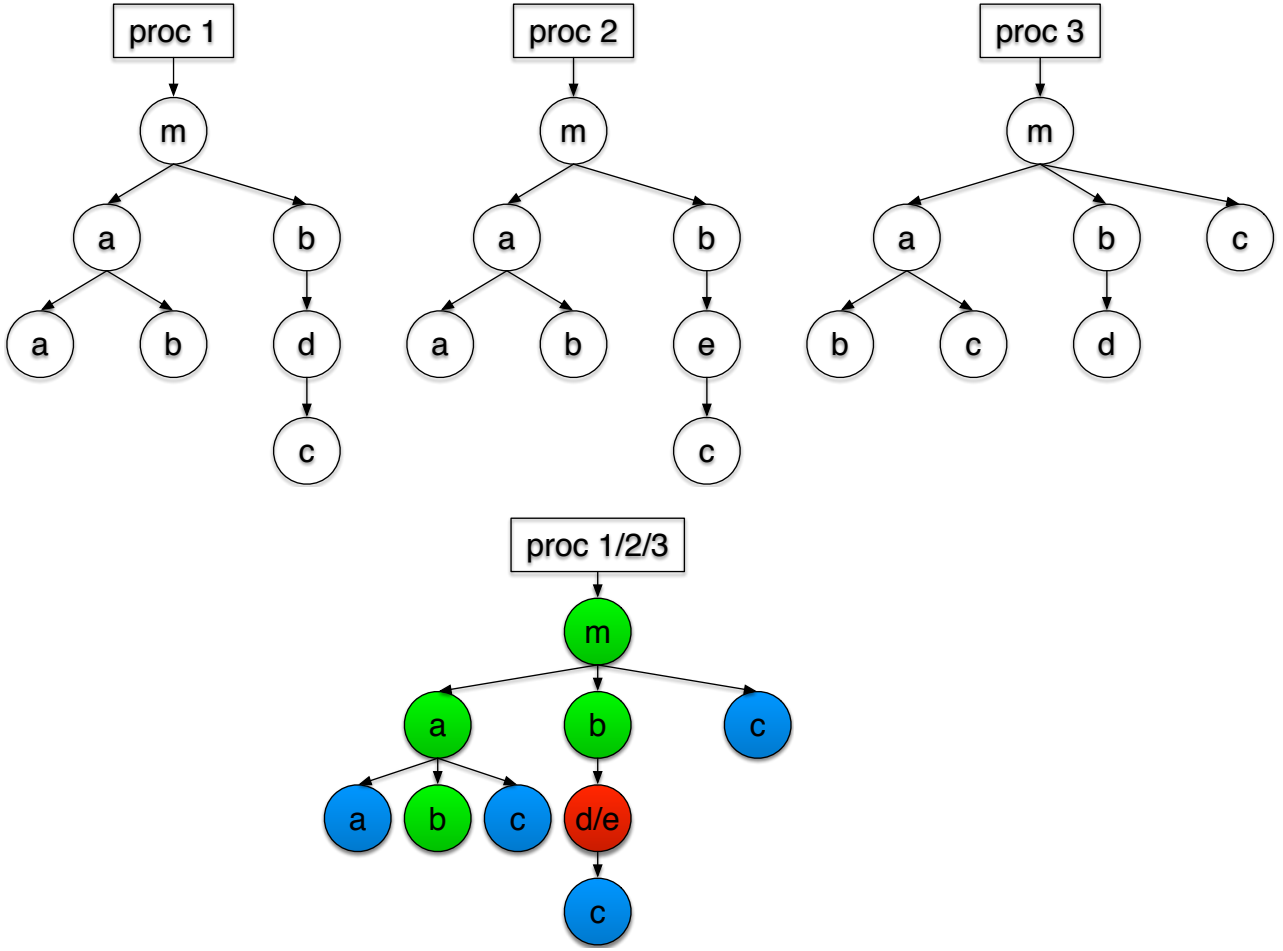
proc 1 a b _

proc 2 a b _

proc 3 _ b c

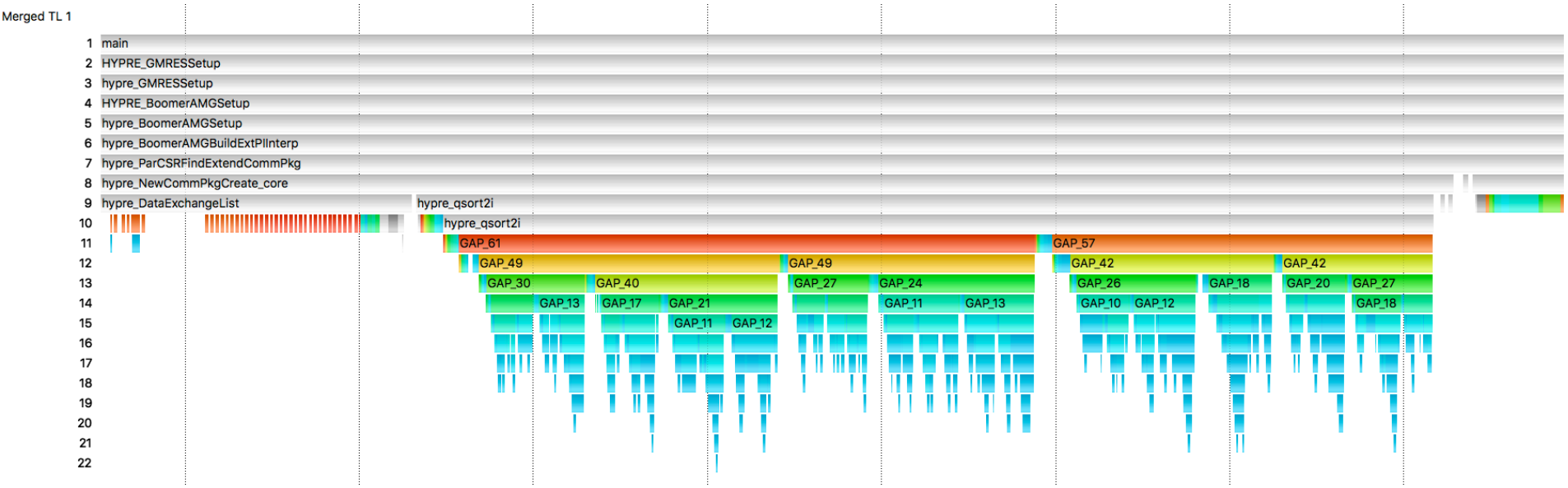
proc 1/2/3

m



Merged Call Graph

- The hierarchical MSA method computes a merged call graph
- The merged call graph:
  - Contains the structural information of all processes
  - Highlights structural similarities and differences
  - Useful for subsequent performance analyses
  - Useful for scalable visualization of performance data



Merged Call Graph Example: AMG2006

- Merged call graph contains information of 64 processes
- White/gray parts are similar between all processes
- Colored areas indicate "missing" processes (GAP states)
- The color indicates the number of processes contained in the function:
  - Red: many processes
  - Blue: few processes

Conclusion

- Introduced a novel grouping method based on the structure of processes
  - Applicable for most application types
  - Grouping can be efficiently stored and computed
  - In most cases linear time complexity
  - In many cases the number of generated clusters remains low and stable for increasing process counts
  - Useful as pre-clustering step to improve improve the effectiveness of traditional analysis techniques
- Introduced a hierarchical multiple sequence alignment approach to compare the structure of processes
  - Compares the function call structure in detail
  - Merged call graph combines the complete structural information of multiple processes and highlights differences

[1] Matthias Weber, Ronny Brendel, and Holger Brunst. Trace File Comparison with a Hierarchical Sequence Alignment Algorithm. ISPA ’12, 2012.

[2] Matthias Weber, Kathryn Mohror, Martin Schulz, Bronis R. de Supinski, Holger Brunst, and Wolfgang E. Nagel. Alignment-Based Metrics for Trace Comparison. Euro-Par’13, 2013.

[3] Matthias Weber, Ronny Brendel, Tobias Hilbrich, Kathryn Mohror, Martin Schulz, and Holger Brunst. Structural Clustering: A New Approach to Support Performance Analysis at Scale. IPDPS, 2016.

[4] Bernhard Ganter and Rudolf Wille. Formal concept analysis, volume 284. Springer Berlin, 1999.

[5] Dean Van Der Merwe, Sergei Obiedkov, and Derrick Kourie. A new incremental algorithm for constructing concept lattices. In Concept Lattices, 2004.