TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Diploma Thesis Presentation

Automatic Profile-Based Characterisation of Performance Traces for Highly Parallel Applications

Ronny Brendel

Professor: Prof. Dr. Wolfgang E. Nagel

Tutors: Matthias Weber, Dr. Holger Brunst

Dresden, 23rd February, 2015

ZIH
Center for Information Services & High Performance Computing

Contents

- Motivation
- Background
  - Software Performance Analysis, Vampir, Challenges
  - Differences in Traces
  - Current Classification Approaches
- Methods
  - A Structural Similarity Measure
  - Storage and Computation of the Measure
  - The Subsumption Measure
- Evaluation
  - Applicability, Scalability
  - The Subsumption Measure
- Conclusion, Future Work, References

Motivation

- Computers pervade everyday life
  - Watches, TVs, cars, washing machines, mobiles,...
- Performance goals
  - Performance analysis required
- Scientific computing relies on parallelism
  - Synchronisation
  - Performance problems
- Parallel programming goes mainstream
  - Eight-core processors in mobile phones

- → Performance analysis and optimisation is more important than ever

[1]

Motivation

- Common software analysis and optimisation workflows **compare** performance data
  - Identify performance issue
  - Implement optimisation
  - Quantify impact of the optimisation

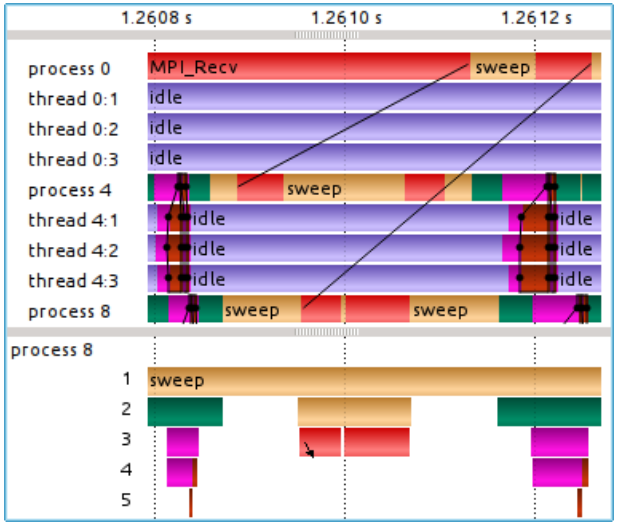
  - Compare different hardware setups
  - Compare timings between processes to detect load imbalance
- Large-scale computers
  - Manually comparing data of thousands of processors is impractical
- → Automatic comparison techniques are needed

Background > Software Performance Analysis

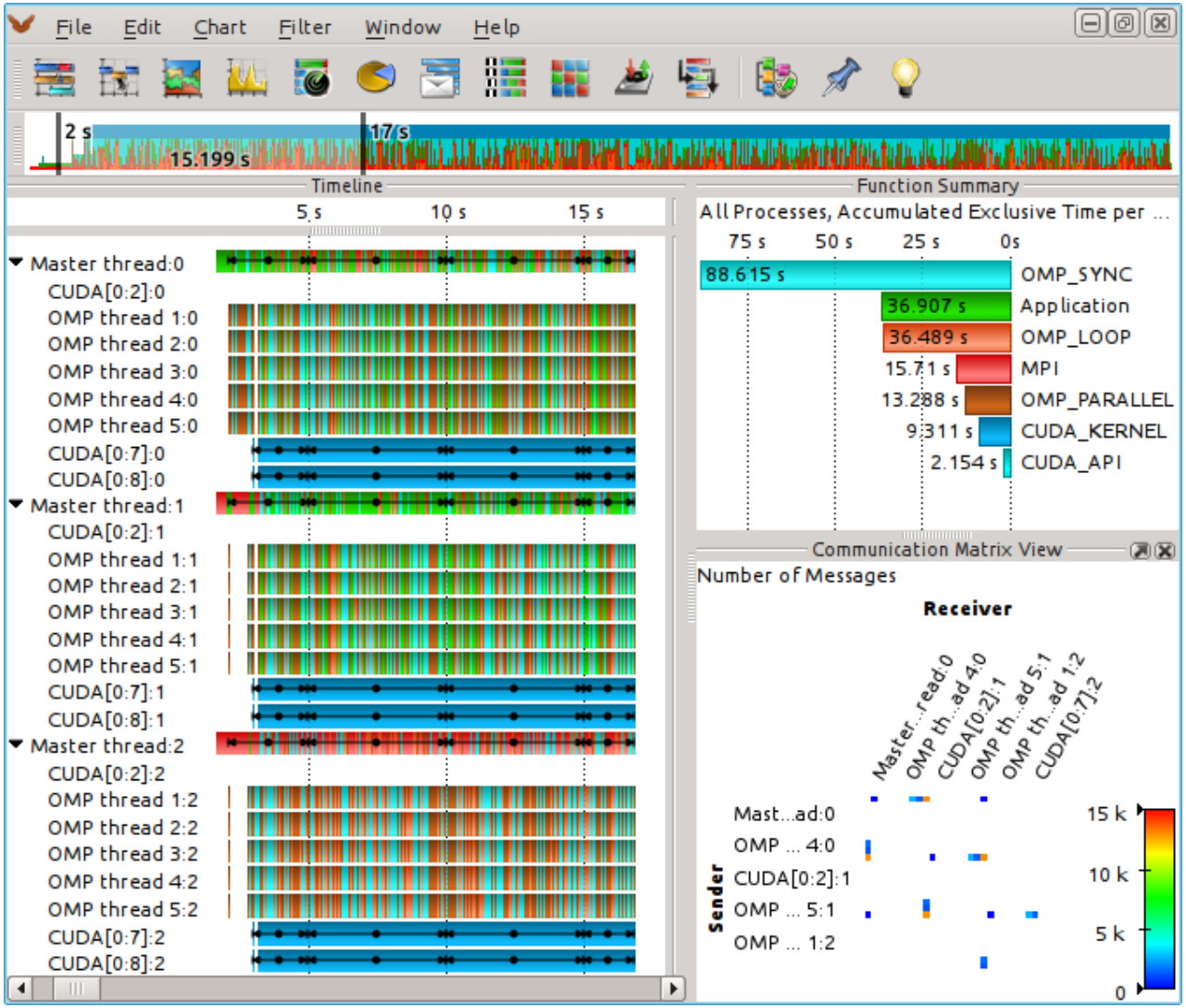
- Profiling
  - Information accumulated per function
  - Sampling
  - Small overhead
  - Static representation

Time (%)	Time (s)	Function name
5.44	1.21	QListData::isEmpty
2.96	0.66	QHash::findNode
2.67	0.60	QList::last
1.71	0.38	handleEnter
0.58	0.13	QHash::find
0.34	0.08	std::forward

- Tracing
  - Detailed information about a program run
  - Instrumentation
  - Large overhead
  - Interactive presentation



Background > Vampir

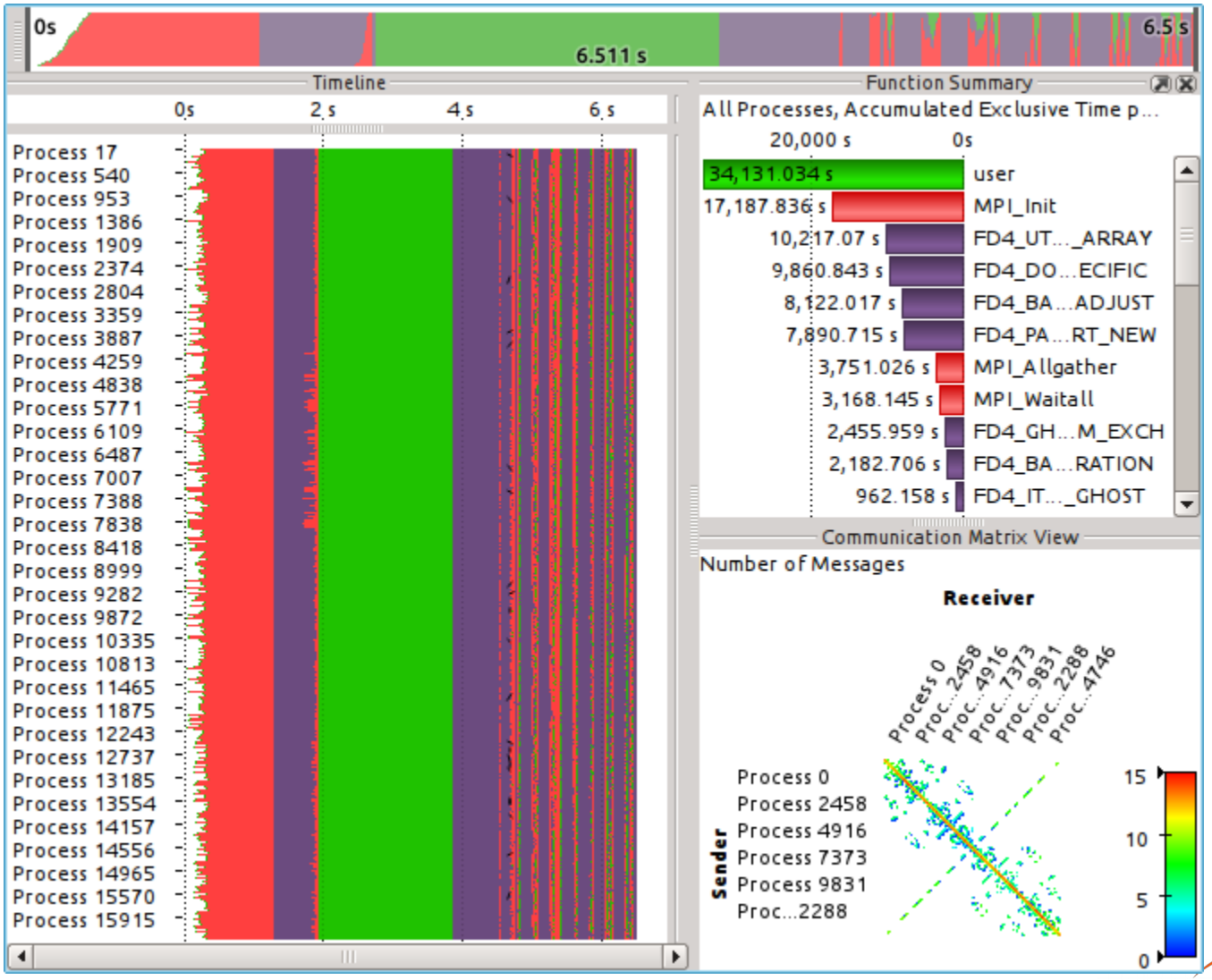


Background > Challenges

- Datastructures & algorithms
  - Achieving scalability with respect to:
    - The number of processes in the trace
    - The number of processes used for analysis
    - The trace file size
  - Limited main memory size

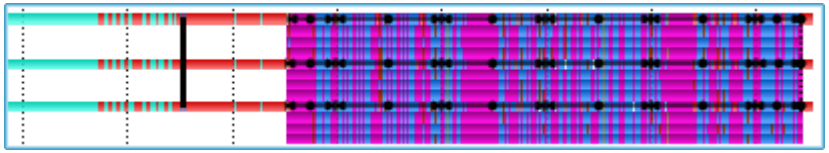
- Visualisation
  - Limited number of pixels on a screen
  - Achieving scalabilty with respect to the number of processes in the trace

Background > Challenges



Background > Differences in Traces

- How does detecting differences between process traces aid performance analysis?
  - Improve scalabity of existing analyses
    - Focus on analysing representatives
    - Preserve screen real estate by e.g. merging similar processes



  - Present new information
    - Visualise timing differences between similar processes
    - Visualise the impact of optimisation
    - Compare runs of the same program on different platforms
  - Aid automatic analysis
    - Detect timing differences between structurally similar processes

Background > Current Classification Approaches

- Computing the similarity for all pairs of processes requires an all-to-all comparison, regardless of the similarity measure
  - Infeasible for 50k+ processes
- Clustering n processes using common clustering algorithms, e.g. k-means and DBSCAN, takes $\mathcal{O}(n^2)$ or more steps
- Most algorithms require a predefined number of clusters and/or make assumptions about the data
  - Leads to unnatural clusters
- → **Scalably** computing a **natural** clustering for a large number of processes requires a different approach

  [2-6]

Methods > A Structural Similarity Measure

- Based on the functions a process invokes

- functions(P) := $\{F \in \mathcal{F} \mid F$ is called on process P at least once$\}$
- functionsimilarity(P_1, P_2)

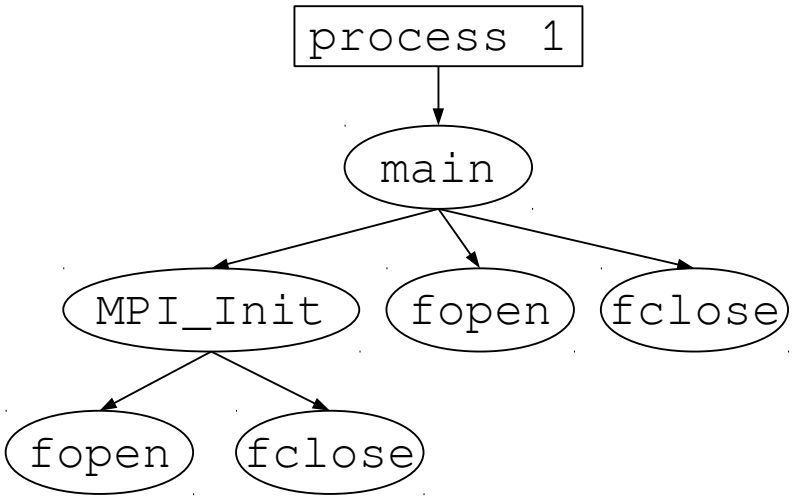
  := |functions(P_1) ∩ functions(P_2)| / |functions(P_1) ∪ functions(P_2)|

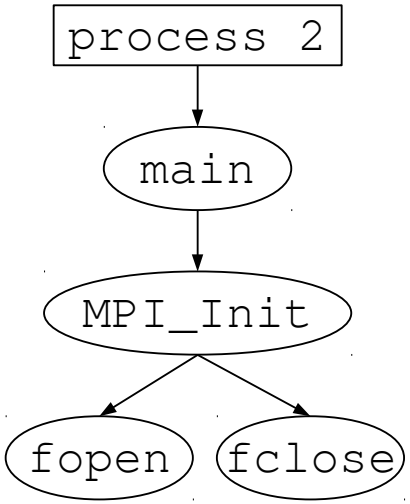
- Alternatively:
  - functionpairs(P) := $\{(F_1, F_2) : F_1$ calls F_2 on P at least once$\}$
  - functionpairsimilarity(P_1, P_2) is analogously defined

[7]

Methods > A Structural Similarity Measure

- Example:
  - Function similarity:
    - functions(p1) = functions(p2) = {ma, MP, fo, fc}
    - |f(p1) ∩ f(p2)| / |f(p1) ∪ f(p2)|
    - 4 / 4 = 100%
  - Function pair similarity:
    - pairs(p1) = {0→ma, ma→MP, ma→fo, ma→fc, MP→fo, MP→fc}
    - pairs(p2) = {0→ma, ma→MP, ma→fo, ma→fc}
    - analogous formula
    - 4 / 6 ≈ 66.6%





Methods > A Structural Similarity Measure

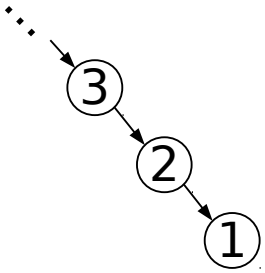
- What influences the measure?
  - Which functions are called ...
  - ... and from which functions
- What does not influence the measure?
  - Number of calls
    - Number of iterations
  - Recursion depth
  - Timing

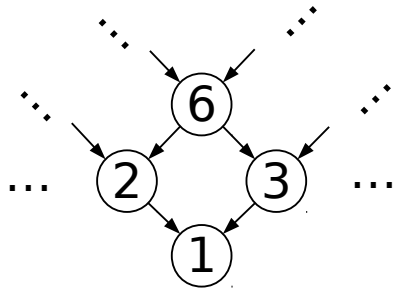
- Assumption:
  - Source code is static → increasing process count or problem size does not increase the number of „equivalence classes“

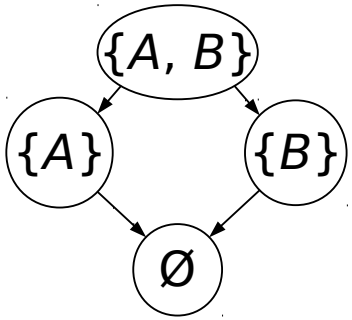
[8-10]

Methods > Storage and Computation of the Measure

- Use a *concept lattice*
  - Total order, e.g. $(\mathbb{N}, \leq)$
  - Partial order, e.g. $(\mathbb{N}, |)$
  - Bounded lattice, e.g. $(\{A, B\}, \cup, \cap, \emptyset, \{A, B\})$



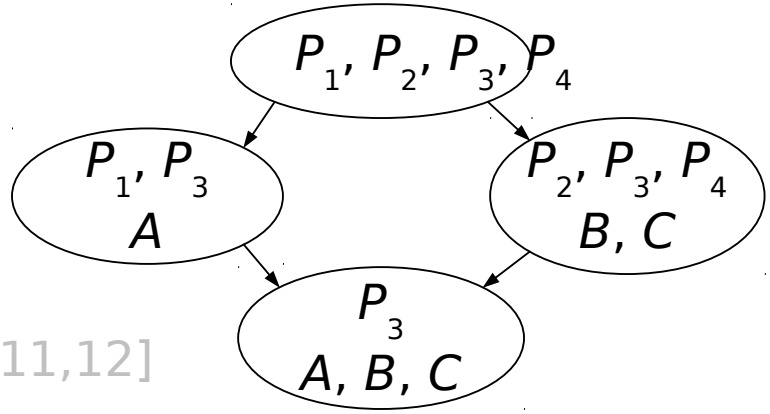


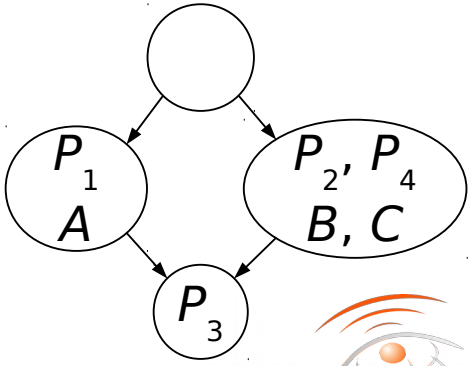


  - Concept lattice $(\mathcal{P}, \mathcal{F}, \mathcal{I} \subseteq \mathcal{P} \times \mathcal{F})$

    where e.g. $\mathcal{P} := \{P_1, P_2, P_3, P_4\}$, $\mathcal{F} := \{A, B, C\}$,

    $\mathcal{I} := \{\ P_1 : A \quad P_2 : B, C \quad P_3 : A, B, C \quad P_4 : B, C\ \}$





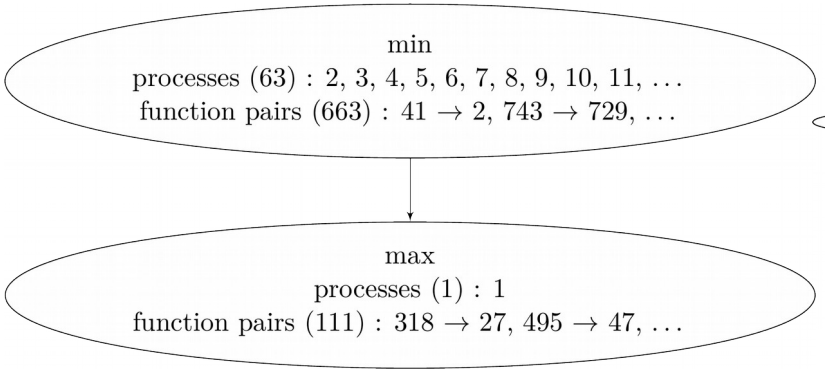
[11,12]

Methods > Storage and Computation of the Measure

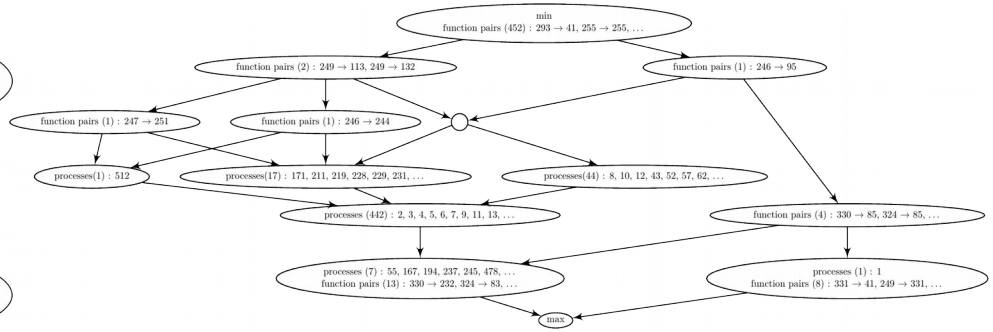
- The result is a directed acyclic graph of „function pair inheritance“

- Expected complexity for building and storing the graph is linear, the worst case is exponential
- Examples:

WRF, 64 processes



AMG, 512 processes

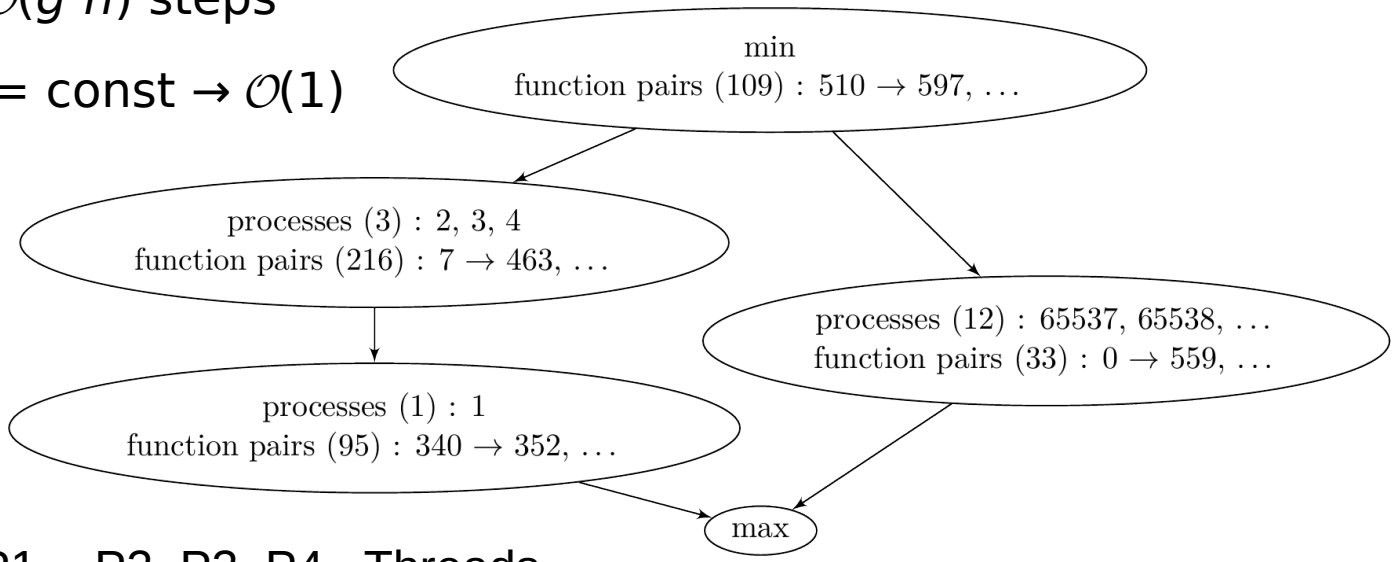


- Next step: compute the similarity between „equivalence classes“

[13,14]

Methods > Storage and Computation of the Measure

- similarity(P_1, P_2) := |pairs(P_1) ∩ pairs(P_2)| / |pairs(P_1) ∪ pairs(P_2)|
- Addition and division of set sizes. No set intersection/union needed.
- For g groups and n nodes, *the* similarity matrix can then be calculated in $\mathcal{O}(g^2 n)$ steps
  - Assume n = const → $\mathcal{O}(1)$
- Example:

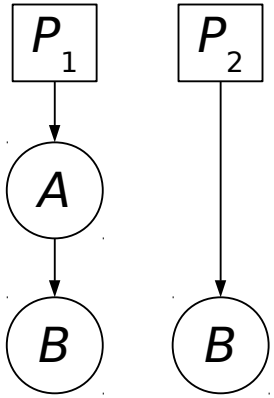


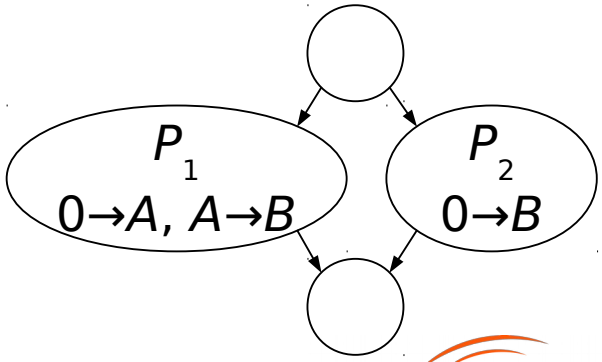
	P1	P2, P3, P4	Threads
P1	1	0.77	0.24
P2, P3, P4	0.77	1	0.30
Threads	0.24	0.30	1

Methods > The Subsumption Measure

- Slightly modifying the techniques enables finding out whether a process calls the same and more functions than another. This way one can detect:
  - Inlining
  - Whether processes are dissimilar because one performs additional calls, e.g. prints debug output or writes to disk
    - Spawned threads
  - Different levels of recording granularity

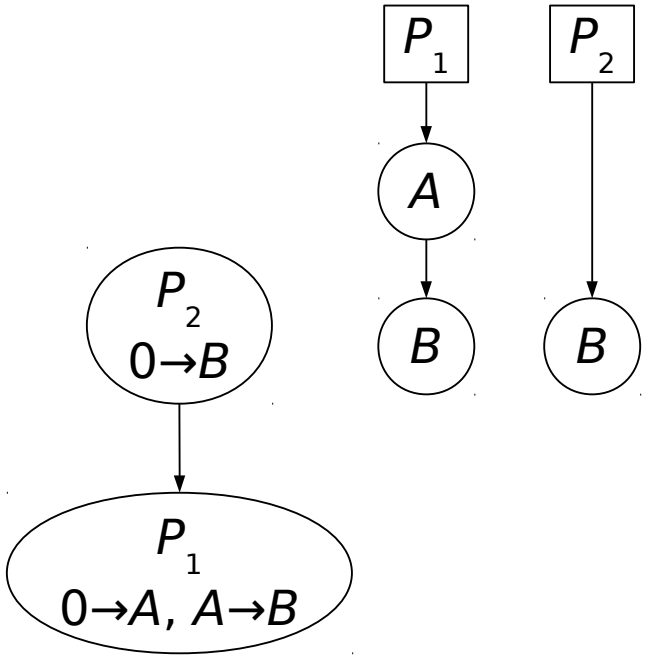
- Example:
  - pairs(P_1) = {0→A, A→B}
  - pairs(P_2) = {0→B}
  - pairsimilarity(P_1, P_2) = 0





Methods > The Subsumption Measure

- 1st step: Calculate transitive closure of both pairs(P_1) and pairs(P_2)
  - pairs(P_1)$^+$ = {0→A, A→B} ∪ {0→B}
  - pairs(P_2)$^+$ = {0→B}



- 2nd step: Generate the concept lattice
- 3rd step: pairsubsumption(P_1, P_2)

  := |pairs(P_1)$^+$ ∩ pairs(P_2)$^+$| / |pairs(P_2)$^+$|

- Example result is 100%

	P_1	P_2
P_1	1	1
P_2	0.33	1

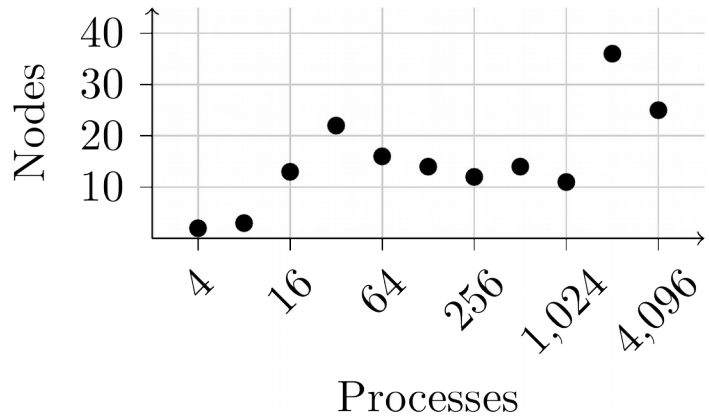
[15]

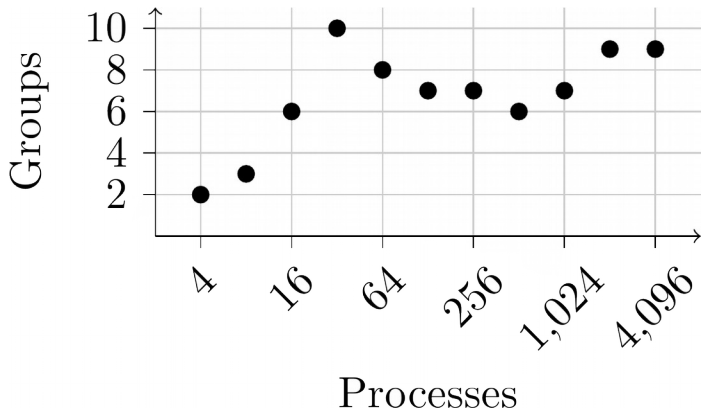
Evaluation > Applicability

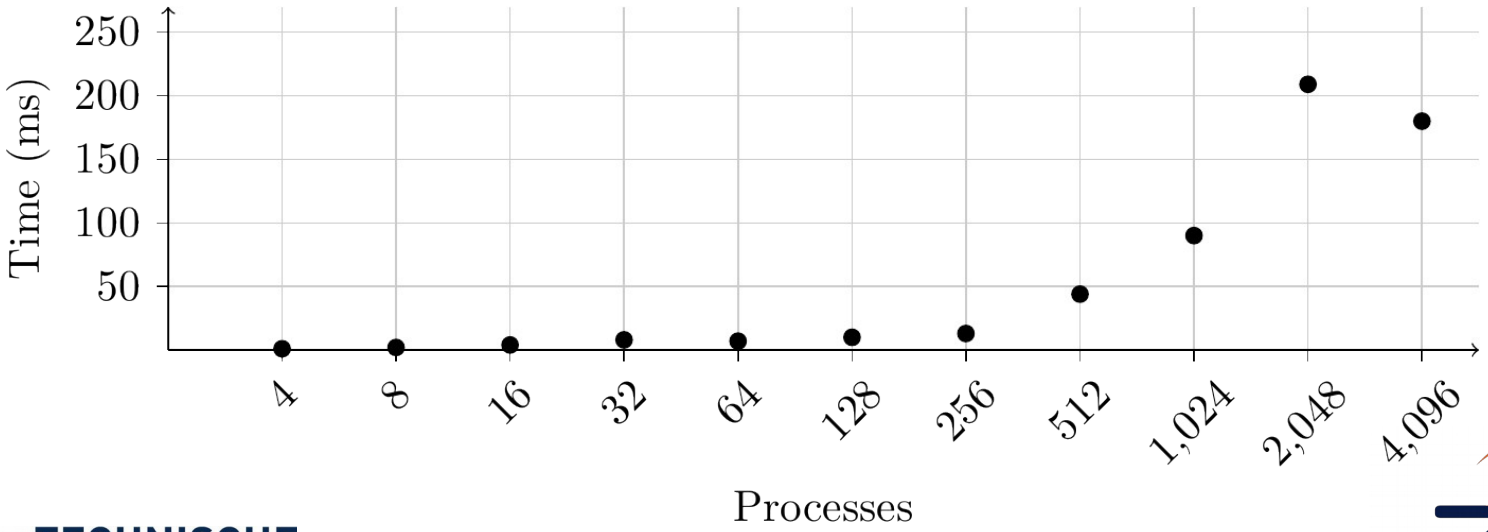
Application	Trace			Result		
	Proc.	~Size (MiB)	Function pairs	t_{eval} (ms)	Nodes	Groups
HPL	2,360	200	8	< 10	2	2
GROMACS	36	3,700	1,381	< 10	24	11
CCLM	180	650	180	< 10	6	3
WRF	64	300	774	< 10	2	2
FD4	65,536	1,000	55	55	22	14
HOMME	1,024	200	179	< 10	3	3
AMG2006	1,024	3,700	440	66	11	7
LULESH	432	350	406	49	182	35
ParaDiS	128	600	649	3,486	6,367	74
PIConGPU	39	300	474	11	60	17
PEPC	16,384	1,200	113	15	2	2

Evaluation > Scalability

- AMG2006: Number of nodes, groups and the timing of concept lattice construction and evaluation

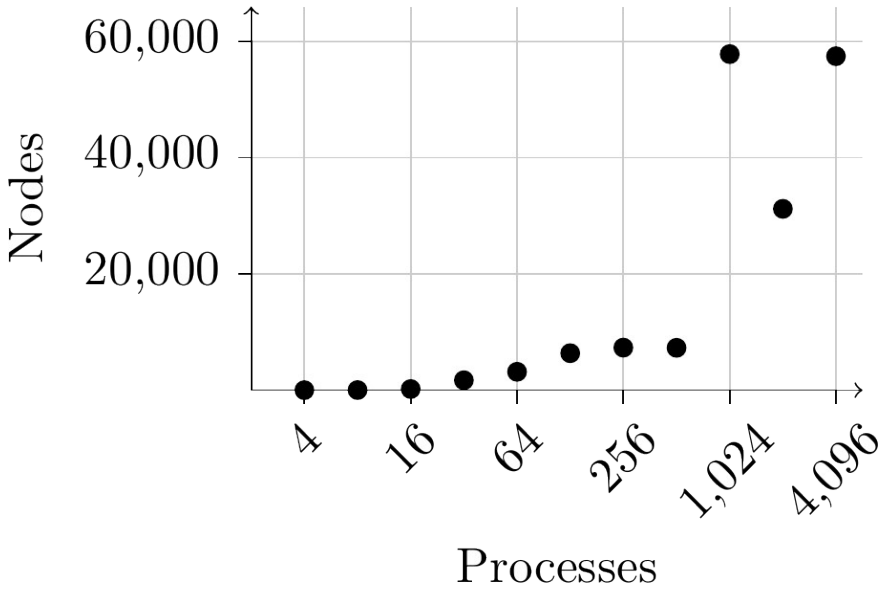


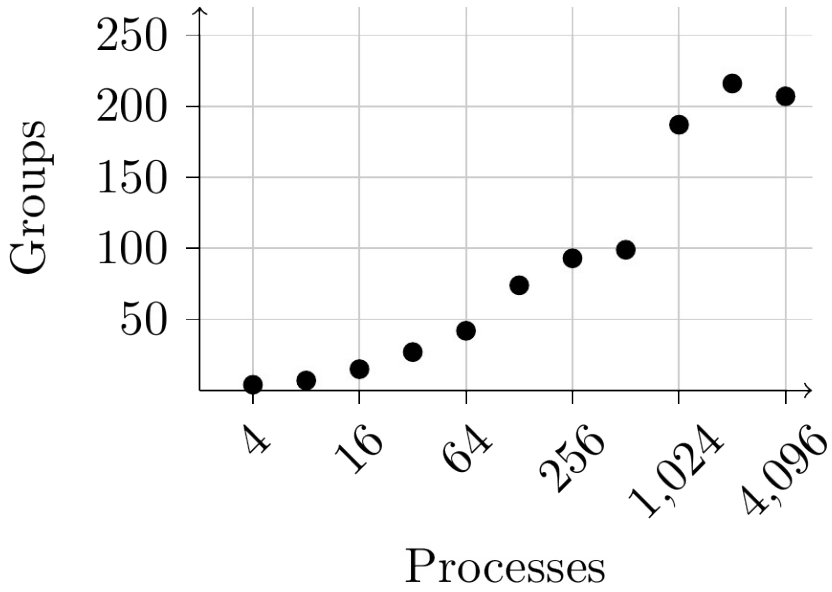




Evaluation > Scalability

- ParaDiS: Number of nodes and groups



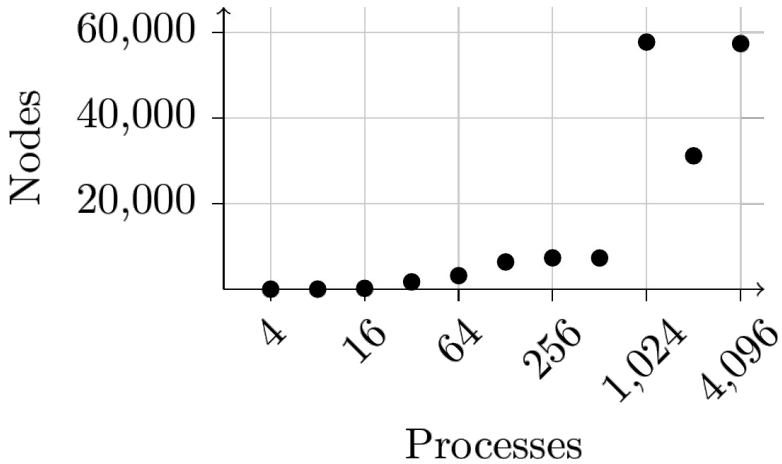


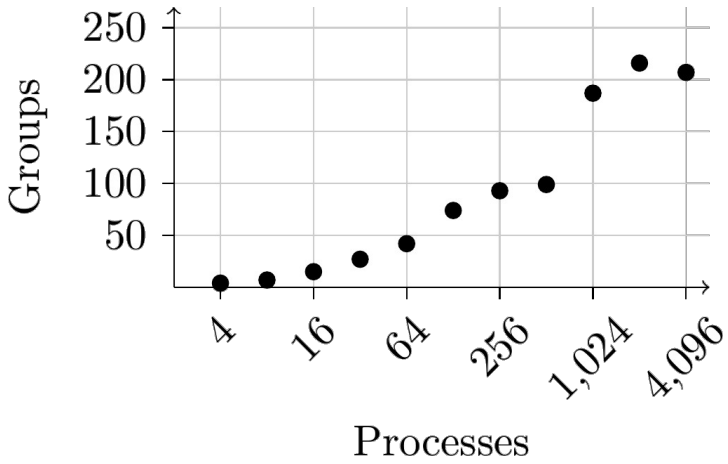
- way too many nodes

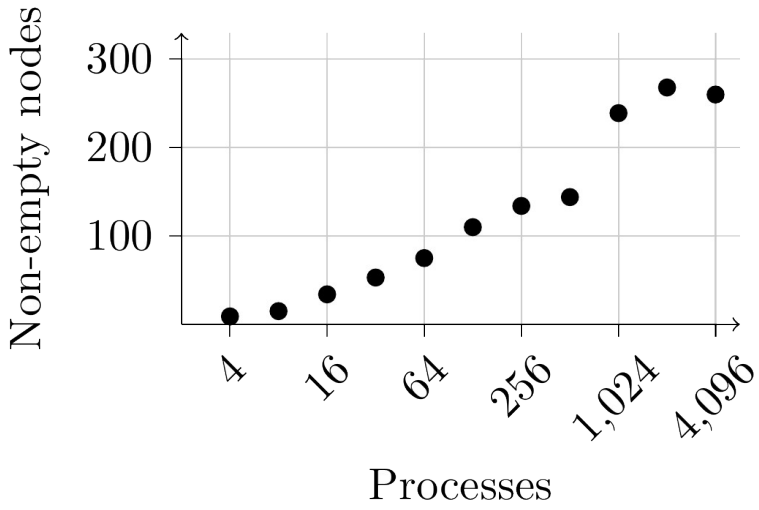
[16]

Evaluation > Scalability

- ParaDiS: Number of nodes, groups and non-empty nodes

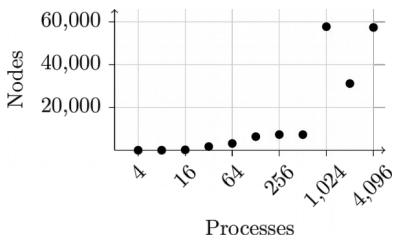


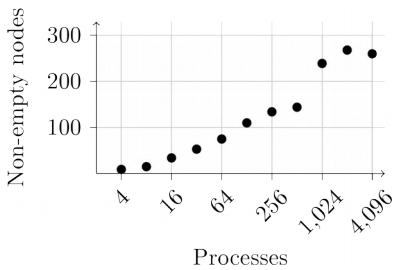


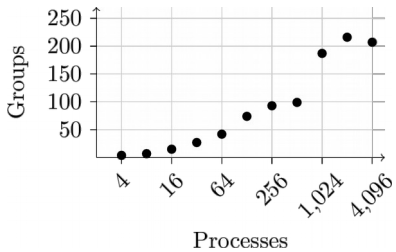


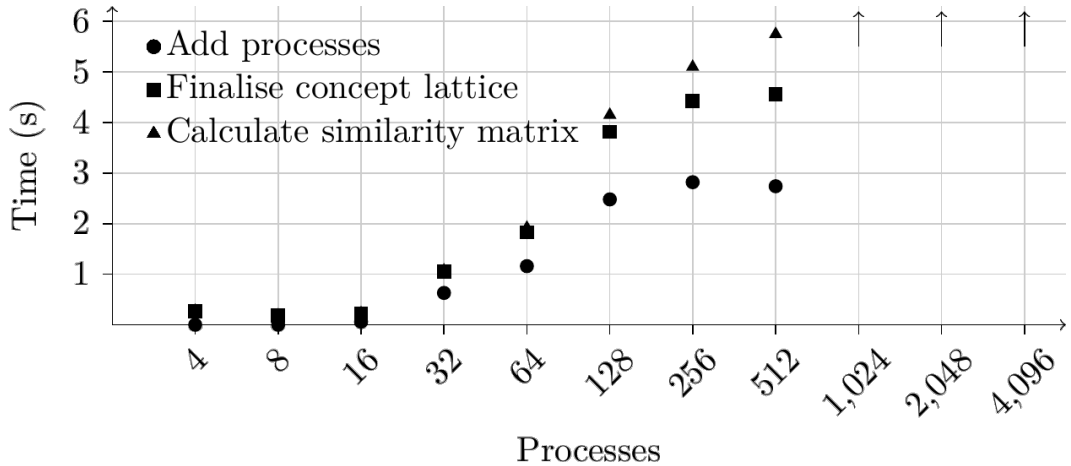
Evaluation > Scalability

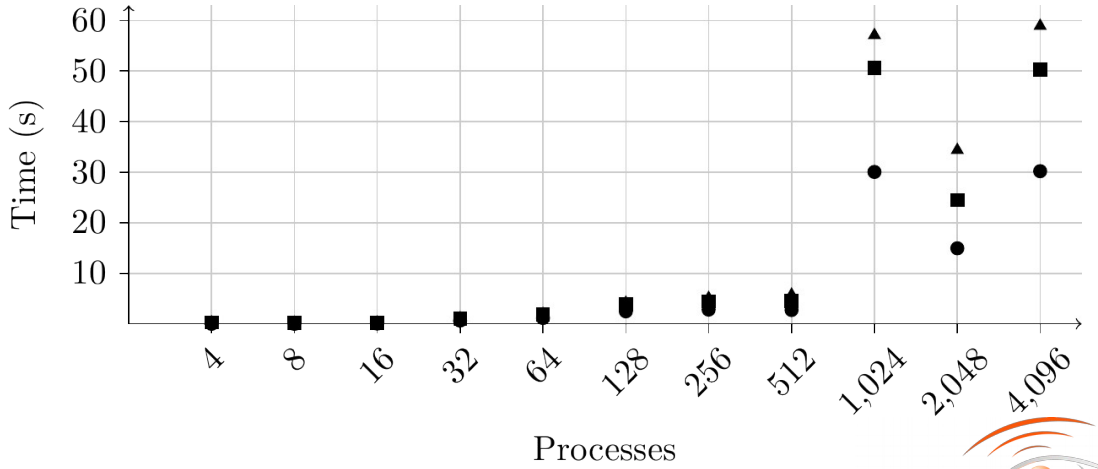
- ParaDiS: Number of nodes, groups and the timing of concept lattice construction and evaluation





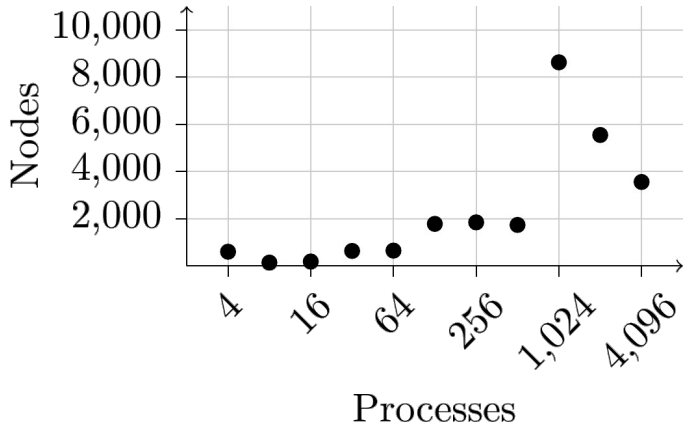


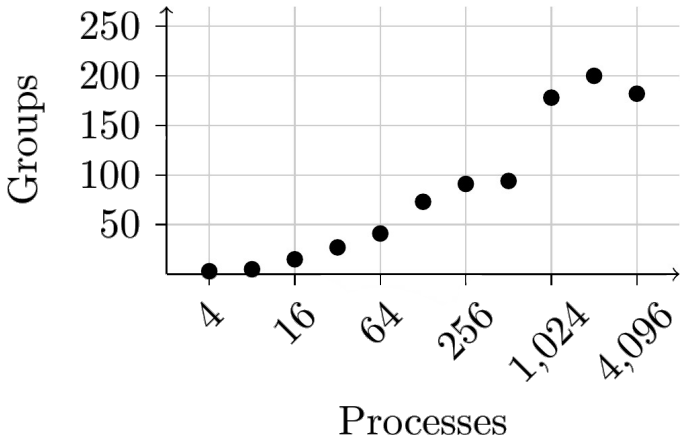


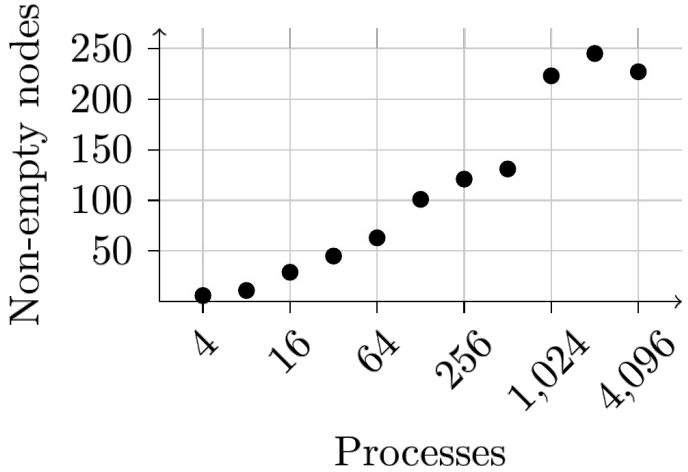


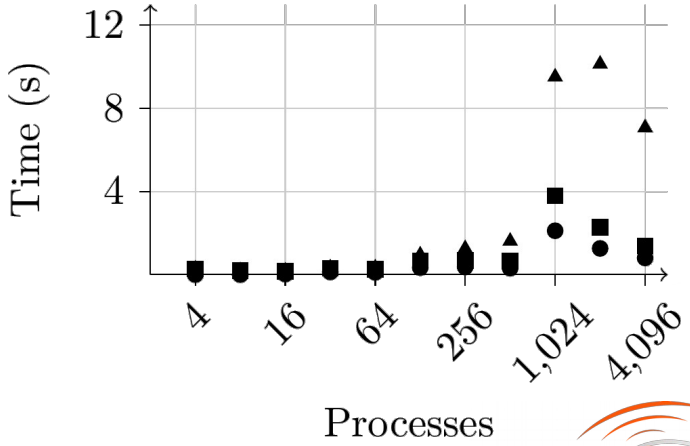
Evaluation > Scalability

- ParaDiS: Functions instead of function pairs; Number of nodes, groups and timing









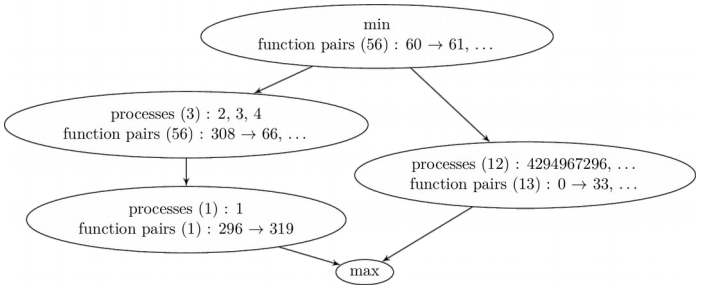
Evaluation > Scalability

- Mostly, programs exhibit regular function call patterns
  - *Single program multiple data*
- Exceptions:
  - Scheduling
    - E.g. CUDA traces with multiple streams
  - Simulation of objects in a space through time
    - Domain decomposition maps cells to processes
    - Objects interact, travel between cells
    - Function calls on a process depend on the objects and their state in the assigned cell
    - E.g. ParaDiS
  - (Dynamic Load Balancing)
- The introduced methods are unsuited for these application types

[8-10]

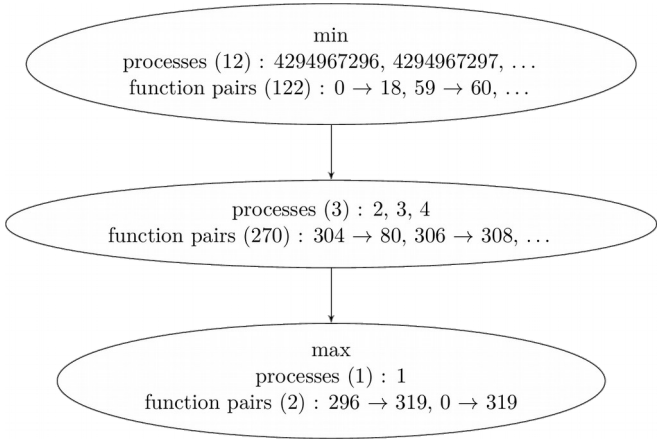
Evaluation > The Subsumption Measure

- BT-MZ, 4 processes, 12 threads
  - Function pair similarity:



	P1	P2, P3, P4	Threads
P1	1	0.99	0.44
P2, P3, P4	0.99	1	0.45
Threads	0.44	0.45	1

  - Function pair subsumption:



	P1	P2, P3, P4	Threads
P1	1	1	1
P2, P3, P4	0.99	1	1
Threads	0.31	0.31	1

[17]

Evaluation > The Subsumption Measure

- GROMACS, different levels of recording granularity

[18]

Evaluation > The Subsumption Measure

- GROMACS, different levels of recording granularity
  - Function pair subsumption:

	MT:0	MT:1	MT:2	MT:3
Master thread:0	1	1	0.63	0.5
Master thread:1	0.73	1	0.63	0.5
Master thread:2	0.05	0.07	1	1
Master thread:3	0.01	0.01	0.24	1

↯ Artifical main function `PARALLEL` introduced for MT:2, MT:3

  - Function subsumption:

	MT:0	MT:1	MT:2	MT:3
Master thread:0	1	1	0.99	0.98
Master thread:1	0.76	1	0.99	0.98
Master thread:2	0.24	0.31	1	1
Master thread:3	0.08	0.10	0.33	1

Conclusion

- Developed methods to cluster processes in $< \mathcal{O}(n^2)$ steps
  - Applicable for most application types
  - Works with many set-based similarity metrics
- Introduced an auxiliary measure to detect a special class of differences
- Demonstrated effectiveness and efficiency of the methods
- Showed how the techniques can be used to improve software performance visualisation using two examples (thesis)

Future Work

- Advance the methods
  - Customise the lattice creation algorithm for our needs
  - Develop mechanisms to reduce the lattice size without losing important function call structure information
- Advance the implementation
  - Parallelisation
  - Fall-backs
  - General optimisation
- Advance software performance visualisation
  - Apply the methods to current visualisations to improve scalability
  - Create new visualisations leveraging the structural similarity information

References

- [1] by GadgetsGuy. http://gadgetsguy.deviantart.com/art/Samsung-Galaxy-S5-by-gadguy-454646219. License: CC-BY-SA 3.0
- [2] Edward W Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.
- [3] Stuart Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.
- [4] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. California, USA, 1967.
- [5] Hugo Steinhaus. Sur la division des corps matériels en parties. *Bulletin de l'Académie Polonaise des Sciences*, 1:801–804, 1956.

References

- [6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.
- [7] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.
- [8] Allinea DDT and MAP user guide — viewing stacks in parallel. https://www.allinea.com/user-guide/DDTControllingProgram Execution.html#x8-9600018, January 2015.
- [9] Dorian C Arnold, Dong H Ahn, Bronis R De Supinski, Gregory L Lee, Barton P Miller, and Martin Schulz. Stack trace analysis for large scale debugging. In *Parallel and Distributed Processing Symposium*, 2007. IPDPS 2007. IEEE International, pages 1–10. IEEE, 2007.
- [10] CW Johnson, PB Thistlewaite, D Walsh, and M Zellner. Developing monitoring and debugging tools for the AP1000 array multiprocessor. In *Proceedings of the Second Fujitsu-ANU CAP Workshop C*, volume 1, page C15, 1991.

References

- [11] Bernhard Ganter and Rudolf Wille. *Formal concept analysis*, volume 284. Springer Berlin, 1999.
- [12] Dean Van Der Merwe, Sergei Obiedkov, and Derrick Kourie. Addintent: A new incremental algorithm for constructing concept lattices. In *Concept Lattices*, pages 372–385. Springer, 2004.
- [13] J Michalakes, S Chen, J Dudhia, L Hart, J Klemp, J Middlecoff, and W Skamarock. Development of a next generation regional weather research and forecast model. In *Developments in Teracomputing: Proceedings of the Ninth ECMWF Workshop on the use of high performance computing in meteorology*, volume 1, pages 269–276. World Scientific, 2001.
- [14] ASC Sequoia benchmark codes – AMG. https://asc.llnl.gov/sequoia/benchmarks/#amg, January 2015.[13]
- [15] Amos Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

References

- [14] Vasily Bulatov, Wei Cai, Jeff Fier, Masato Hiratani, Gregg Hommes, Tim Pierce, Meijie Tang, Moono Rhee, Kim Yates, and Tom Arsenlis. Scalable line dynamics in ParaDiS. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 19. IEEE Computer Society, 2004.
- [17] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Russell L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The NAS parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.
- [18] David Van Der Spoel, Erik Lindahl, Berk Hess, Gerrit Groenhof, Alan E Mark, and Herman JC Berendsen. GROMACS: fast, flexible, and free. *Journal of computational chemistry*, 26(16):1701–1718, 2005.

References

- [19] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S. Müller, and Wolfgang E. Nagel. The Vampir Performance Analysis Tool-Set. In *Tools for High Performance Computing*, pages 139–155. Springer, 2008.

Thank You!