TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Interim Diploma Presentation

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

Ronny Brendel

Professor: Prof. Dr. Wolfgang E. Nagel

Tutors: Matthias Weber, Dr. Holger Brunst

Dresden, 3rd September, 2014

ZIH
Zentrum für Informationsdienste
und Hochleistungsrechnen

Contents

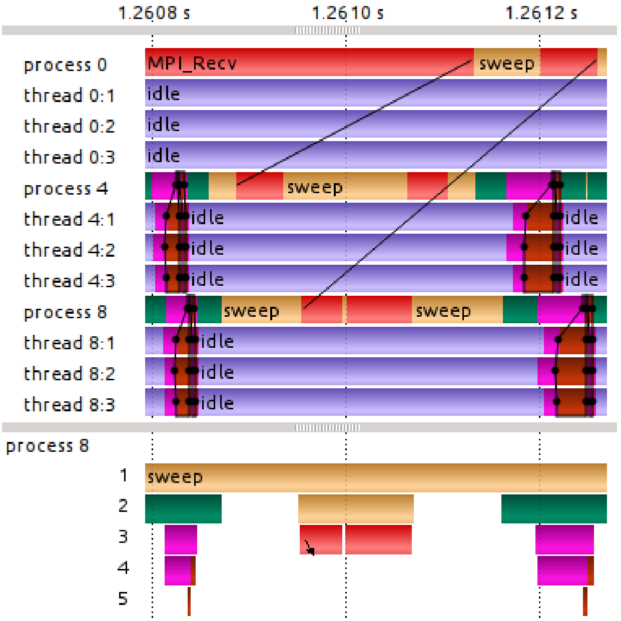
- Introduction
  - Performance Analysis, Vampir, Challenges
  - Differences in Traces
- Classifying Process Traces
  - Current Approaches
  - Methodology
  - Evaluation
- A Novel Profile Display
  - State of the Art
  - Outlook
- Conclusion, Future Work, References

Introduction > Performance Analysis

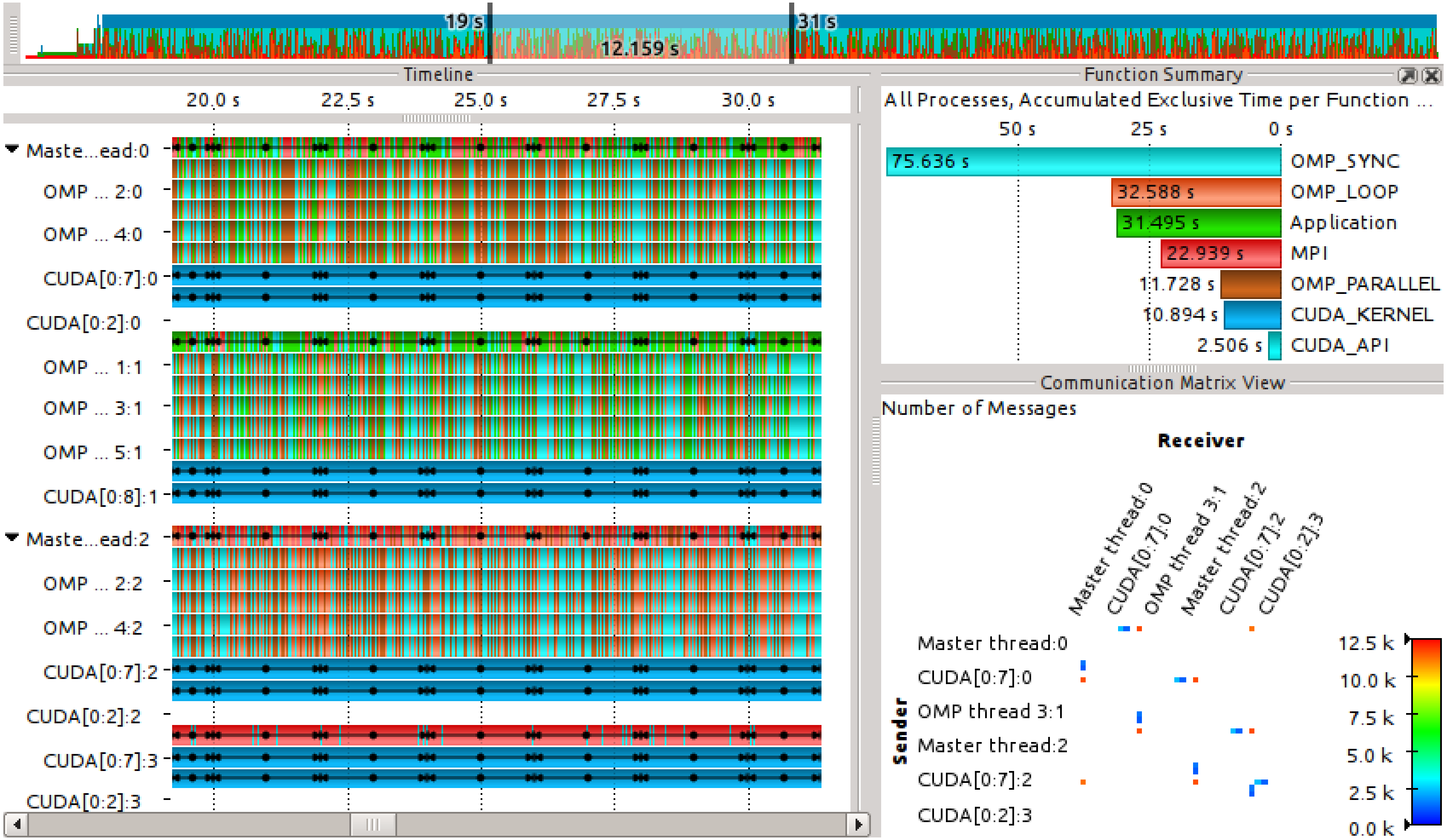
- Profiling
  - Information accumulated per function and process
  - Sampled
  - Small overhead

```
cumulative   self
seconds    seconds calls name
    0.31      0.05 25195 QList::isEmpty()
    0.40      0.04 30239 QList::Node::t()
    0.44      0.04 12294 QList::end()
    0.55      0.03  3696 QList::end()
    0.70      0.03  4939 handleEnter
    0.73      0.03 36939 handleLeave
    0.88      0.02 99207 void std::swap()
```

- Tracing
  - Complete information about a program run
  - Instrumented
  - Large overhead



Introduction > Vampir

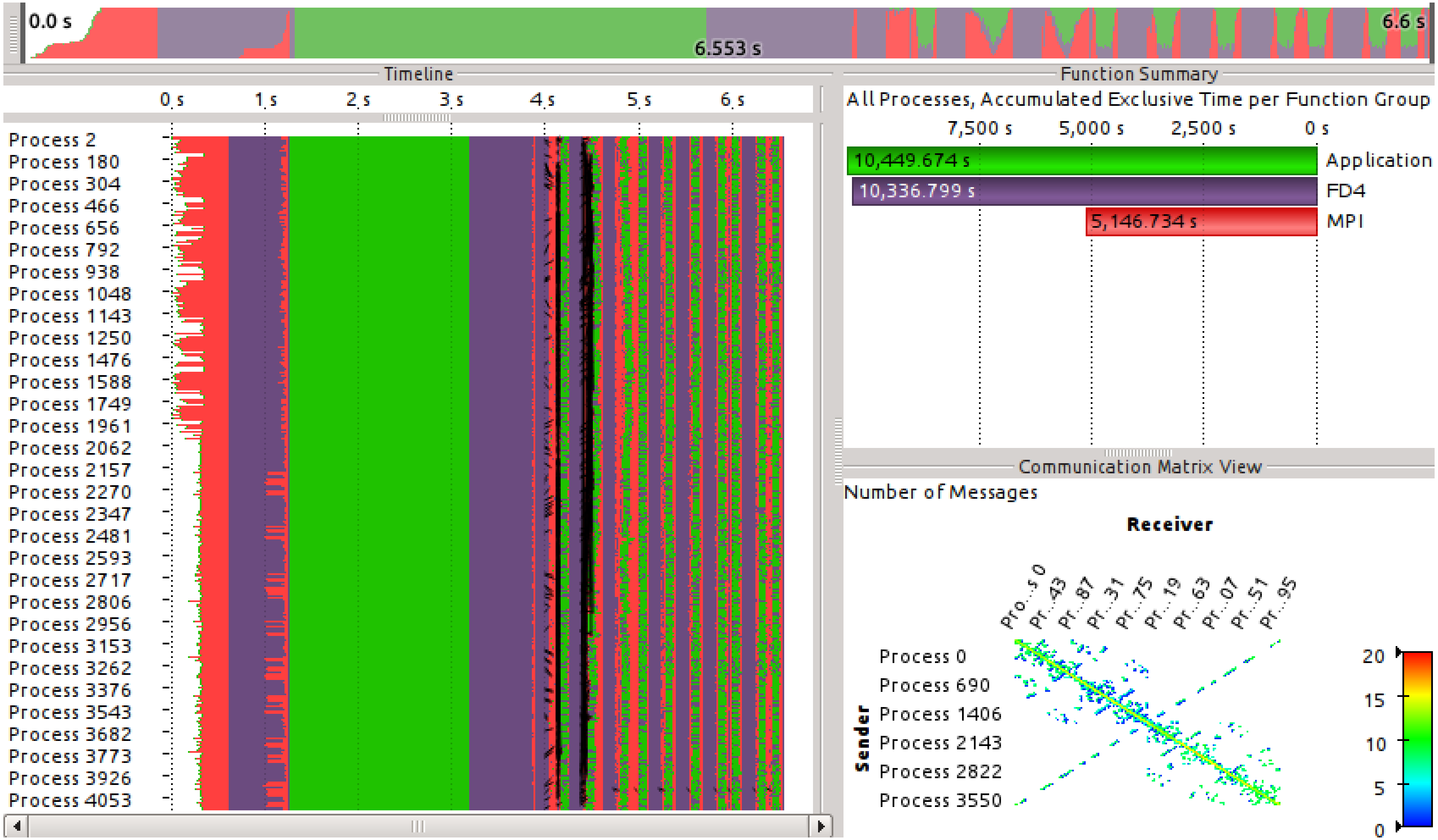


Introduction > Challenges

- Datastructures & algorithms:
  - Limited main memory size
  - Achieving scalibility wrt:
    - Number of processes in the trace
    - Number of processes used for analysis
    - Tracefile Size

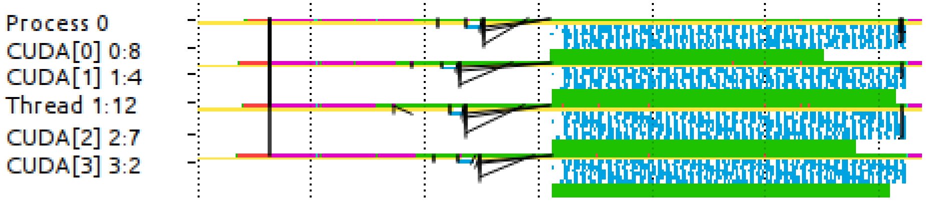
- Visualisation:
  - Limited number of Pixels on a screen
  - Achieving scalibilty wrt the number of processes in the trace

Introduction > Challenges



Introduction > Differences in Traces

- How does detecting differences between process traces aid performance analysis?
  - Present new information
    - Visualise timing differences between similar processes
    - Visualise the impact of optimisation
    - Compare runs of the same program on different platforms
  - Improve scalibity of existing views
    - Preserve screen real estate by e.g. merging similar processes



  - Aid automatic analysis
    - Detect timing differences between structurally similar processes

Classifying Process Traces > Current Approaches

- Take any similarity measure. Calculating it for all pairs of processes already requires an all-to-all comparison, which is not feasible for 50k+ processes.
- Clustering n processes using common clustering algorithms, e.g. k-means and DBSCAN, takes at least n^2 steps
- Some algorithms require a predefined number of clusters or make assumptions about the data layout, which leads to unnatural clusters.
- To handle large amounts of processes properly, we need something better

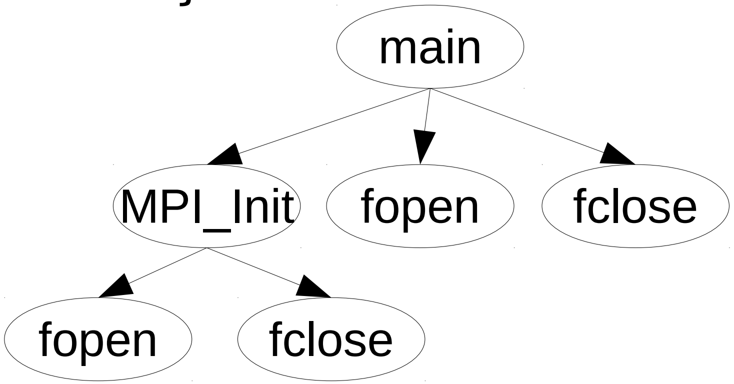
Classifying Process Traces > Methodology

- A simple, structural similarity metric
  - Each process p is assigned a set of functions

    {A | A is called on p at least once}
  - Similarity(P1, P2) := |P1∩P2| / |P1∪P2|

  - Alternatively: each process p is assigned a set of function pairs

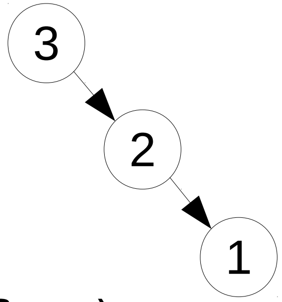
    {(A, B) : A calls B on process p at least once}
  - Similarity is calculated analogously

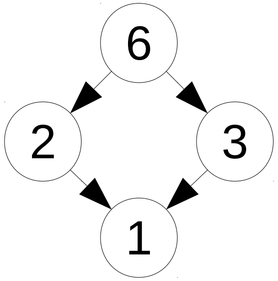


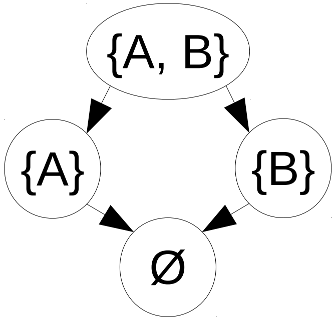
- Next step: Calculate and store similarity between processes

Classifying Process Traces > Methodology

- Use a concept lattice
  - Total order, e.g. (ℕ, ≤)
  - Partial order, e.g. (ℕ, |)
  - Bounded lattice, e.g. ({A, B}, ⊆)



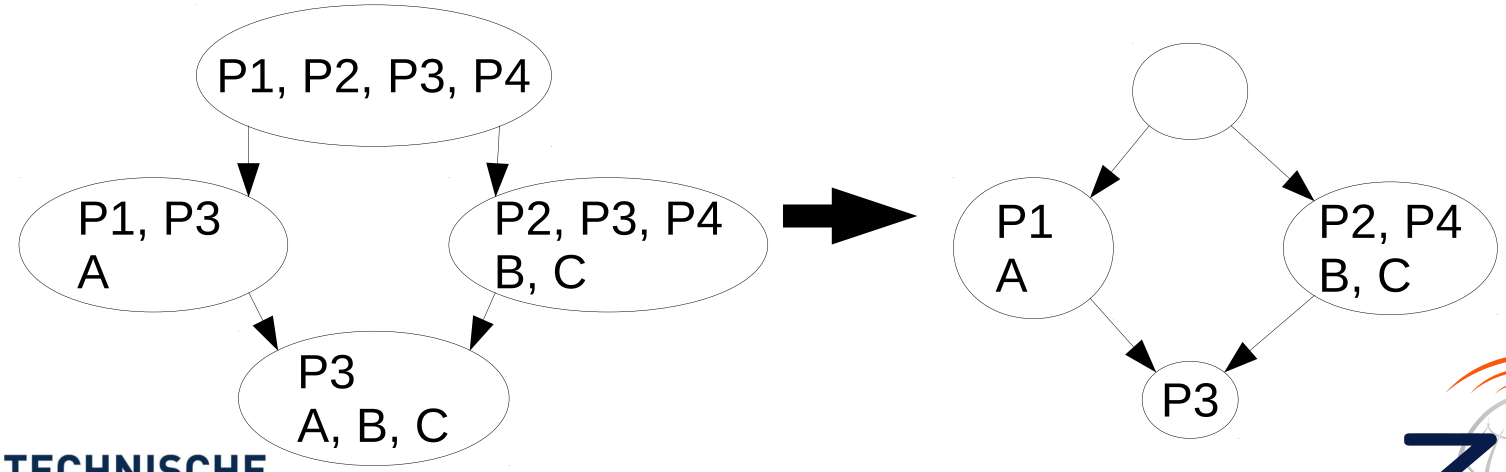




  - Concept lattice $(\mathcal{P}, \mathcal{F}, \mathcal{I} \subseteq \mathcal{P} \times \mathcal{F})$

    where e.g. $\mathcal{P}$:= {P1, P2, P3}, $\mathcal{F}$:= {A, B, C},

    $\mathcal{I}$:= { P1 : A P2 : B, C P3 : A, B, C P4: B, C }

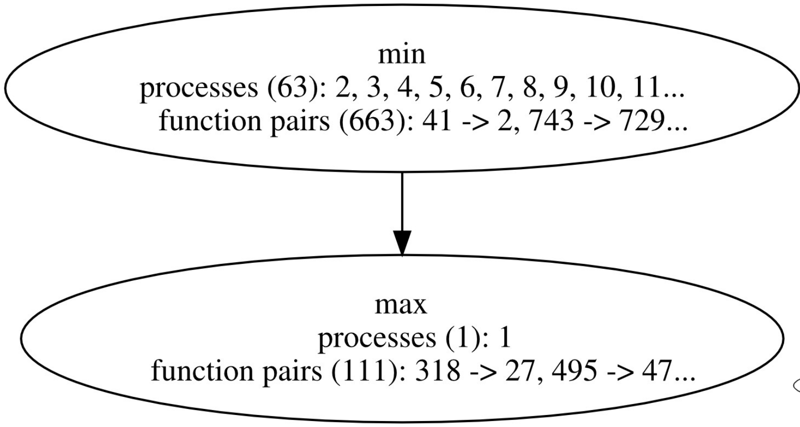


Classifying Process Traces > Methodology

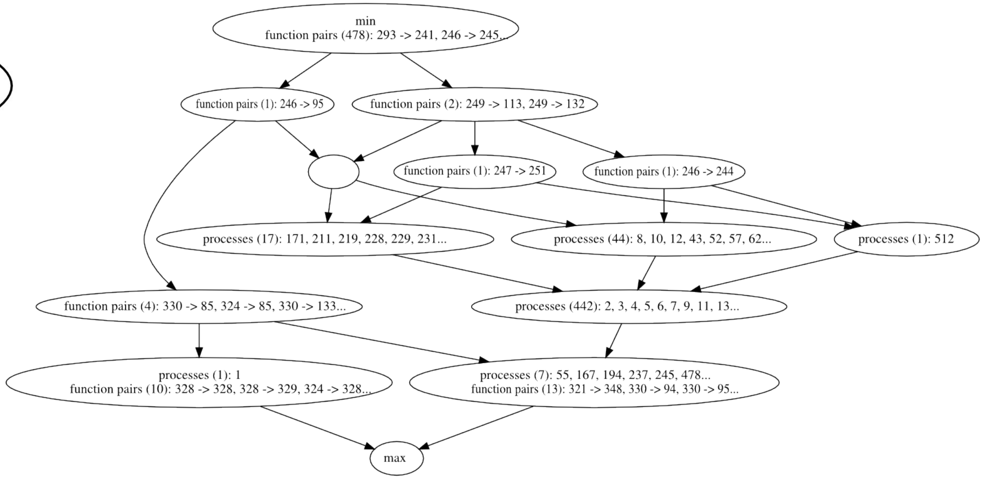
- The result is a directed acyclic graph of „function pair inheritance“

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

WRF 64 processes


min
processes (63): 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
function pairs (663): 41 -> 2, 743 -> 729...
max
processes (1): 1
function pairs (111): 318 -> 27, 495 -> 47...

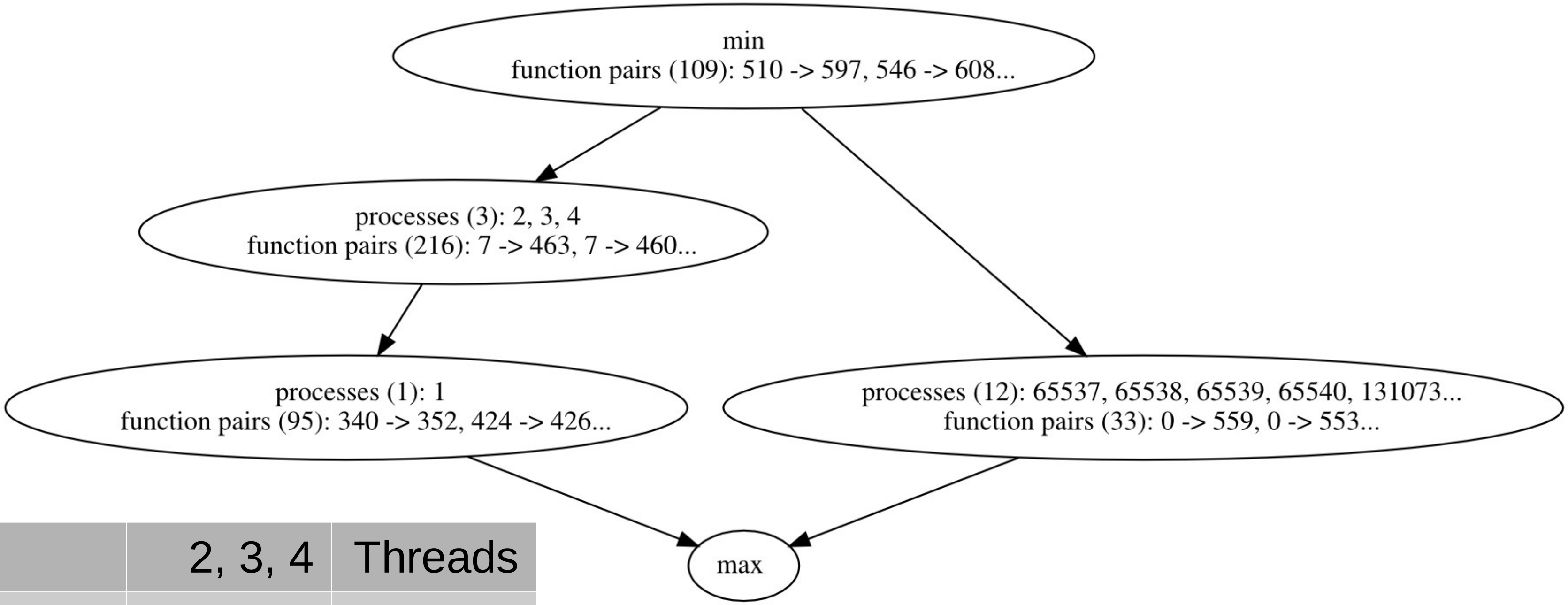
AMG 512 processes


min
function pairs (478): 293 -> 241, 246 -> 245...
function pairs (1): 246 -> 95
function pairs (2): 249 -> 113, 249 -> 132
function pairs (1): 247 -> 251
function pairs (1): 246 -> 244
processes (17): 171, 211, 219, 228, 229, 231...
processes (44): 8, 10, 12, 43, 52, 57, 62...
processes (1): 512
function pairs (4): 330 -> 85, 324 -> 85, 330 -> 133...
processes (442): 2, 3, 4, 5, 6, 7, 9, 11, 13...
processes (1): 1
function pairs (10): 328 -> 328, 328 -> 329, 324 -> 328...
processes (7): 55, 167, 194, 237, 245, 478...
function pairs (13): 321 -> 348, 330 -> 94, 330 -> 95...
max

- Next step: calculate similarity between „equivalence classes“

Classifying Process Traces > Methodology

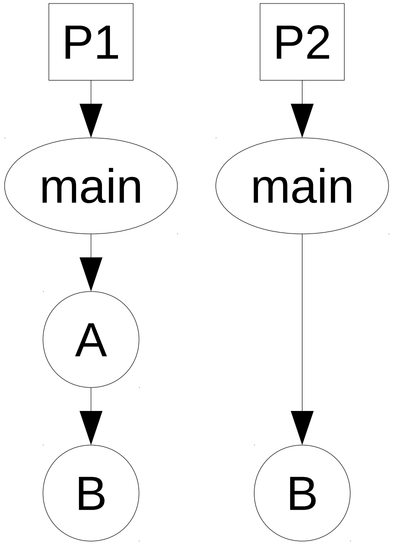
- Similarity(P1, P2) := |P1∩P2| / |P1∪P2|
  - Simple addition and division of set sizes. No set operations involved.
  - Similarity matrix can then be calculated in $O(g^2n)$ steps


min
function pairs (109): 510 -> 597, 546 -> 608...
processes (3): 2, 3, 4
function pairs (216): 7 -> 463, 7 -> 460...
processes (1): 1
function pairs (95): 340 -> 352, 424 -> 426...
processes (12): 65537, 65538, 65539, 65540, 131073...
function pairs (33): 0 -> 559, 0 -> 553...
max

	2, 3, 4	Threads
1	77%	24%
2, 3, 4		30%

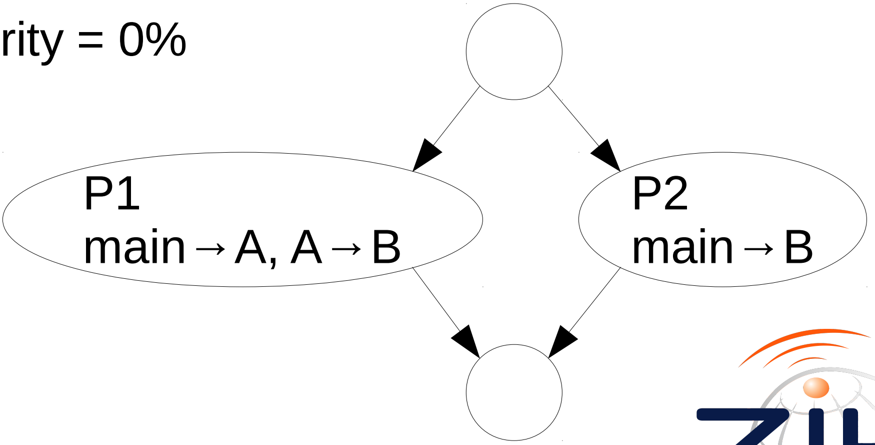
Classifying Process Traces > Methodology

- Slightly modifying this technique lets us find out whether a process calls the same and more functions than another. This way we can:
  - Detect inlining
  - Detect wether processes are dissimilar because one performs additional calls, e.g. logs output
  - Detect different levels of coarsness of trace recording



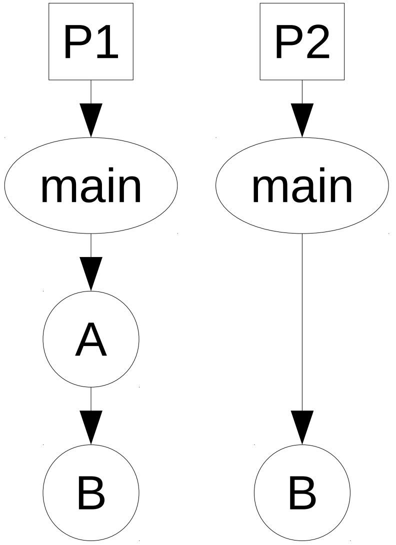
P1 = {(main, A), (A, B)}
P2 = {(main, B)}

Similarity = 0%



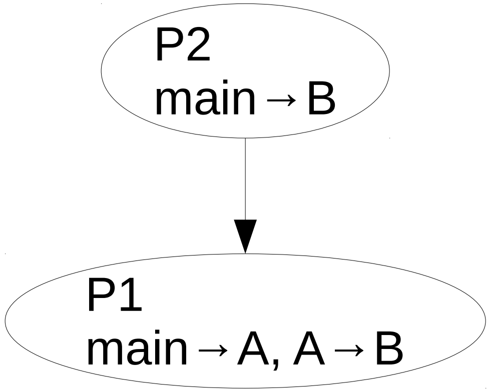
Classifying Process Traces > Methodology

- 1st step: Calculate transitive closure of both P and Q



P1 = {(main, A), (A, B)} ∪ {(main, B)}
P2 = {(main, B)}

- 2nd step: Build the concept lattice



- 3rd step: SubsumptionMetric(P1, P2) := |P1∩P2| / |P2|
- Example result is 100%

	P1	P2
P1		1
P2	1/3	

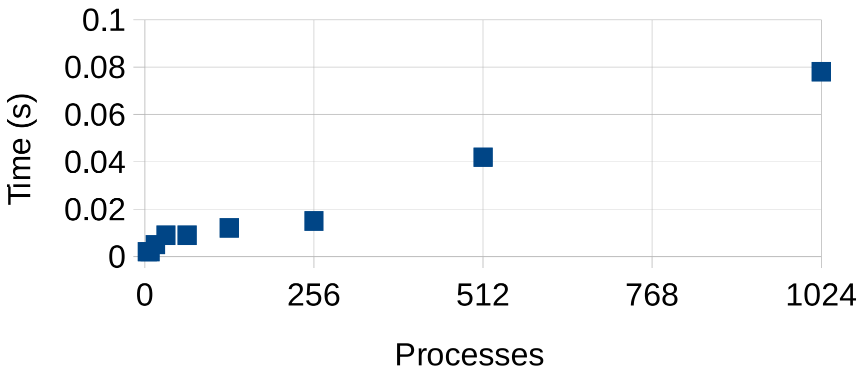
Classifying Process Traces > Evaluation

Trace			Result		
Application	**Proc.**	**Size (MiB)**	**Construction Time (%)**	**Nodes**	**Groups**
LINPACK	832	145	≤ 0.1	3	3
Gromacs	36	3,700	≤ 0.1	24	11
COSMO-SPECS	100	976	≤ 0.1	1	1
FD4	4096	166	≤ 0.1	2	2
WRF	64	283	≤ 0.1	2	2
HOMME	1024	179	≤ 0.1	3	3
LULESH	432	354	0.27	182	35
PIConGPU	39	286	≤ 0.1	60	17
BT	16	150	≤ 0.1	5	3

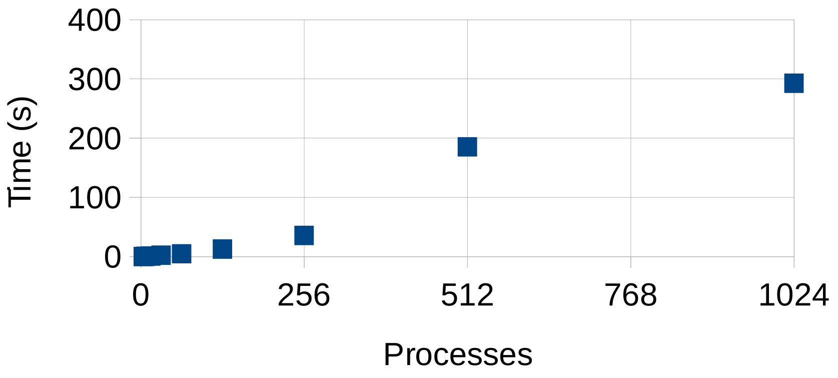
Classifying Process Traces > Evaluation

- AMG2006

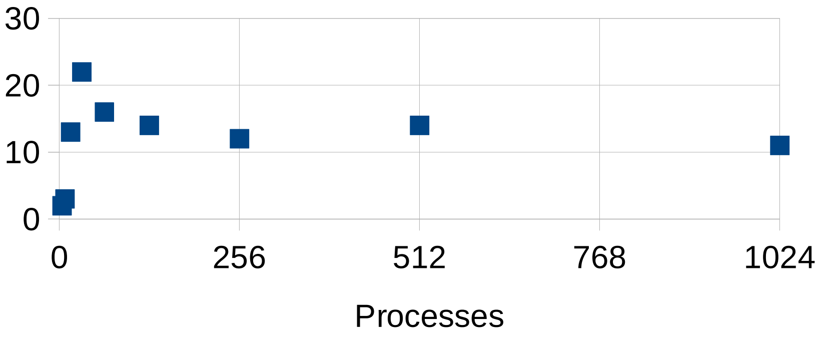
Construction Time



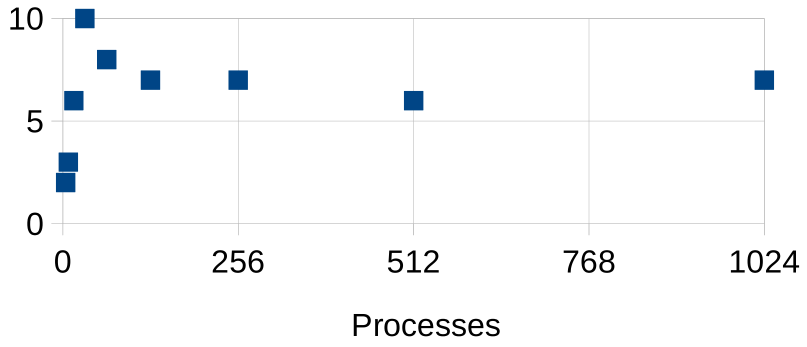
Tracefile Loading Time



Number of Nodes in the Lattice

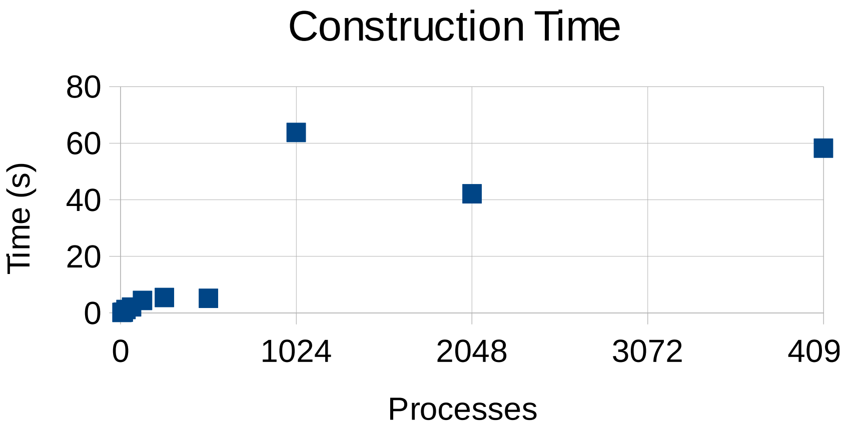


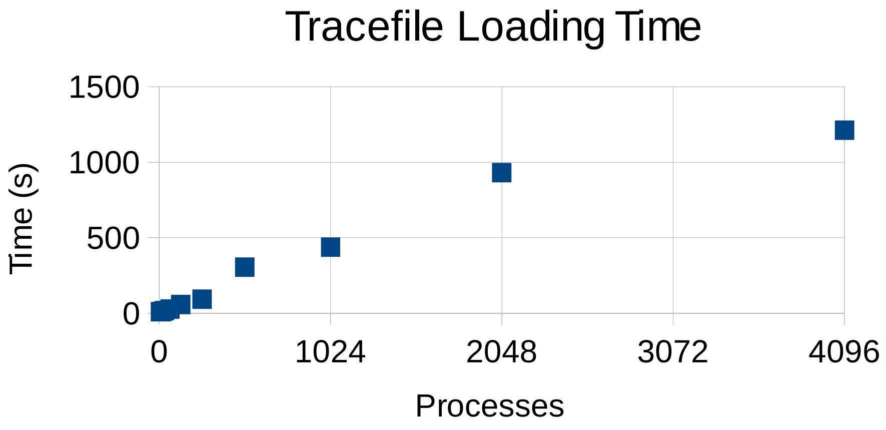
Number of Groups

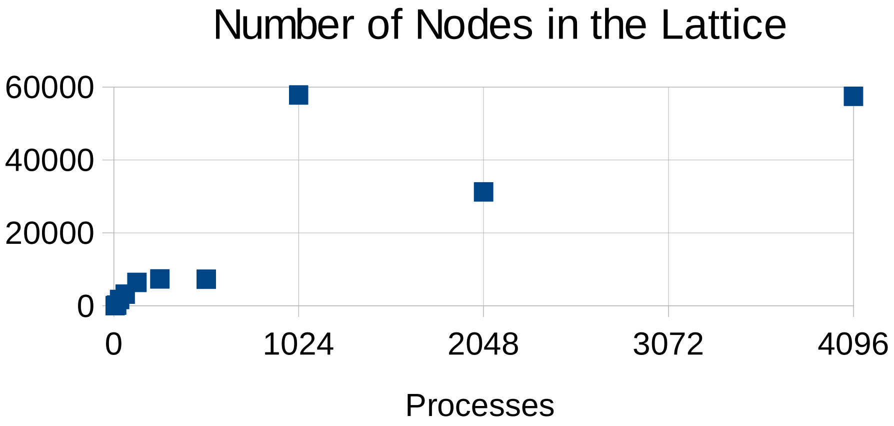


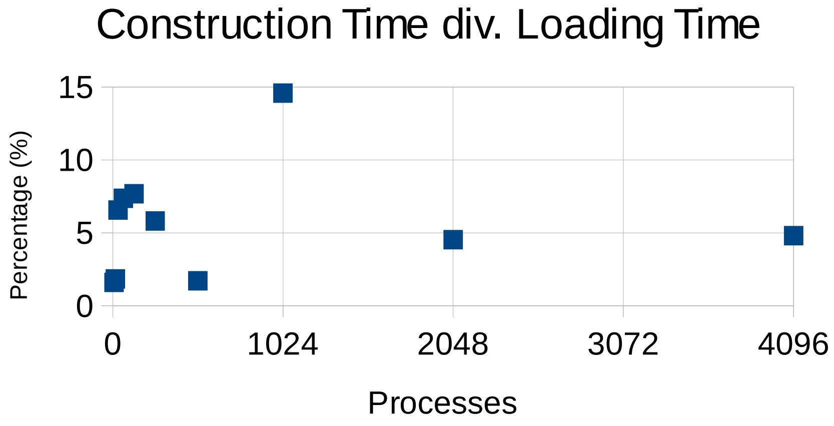
Classifying Process Traces > Evaluation

- ParaDiS


Construction Time

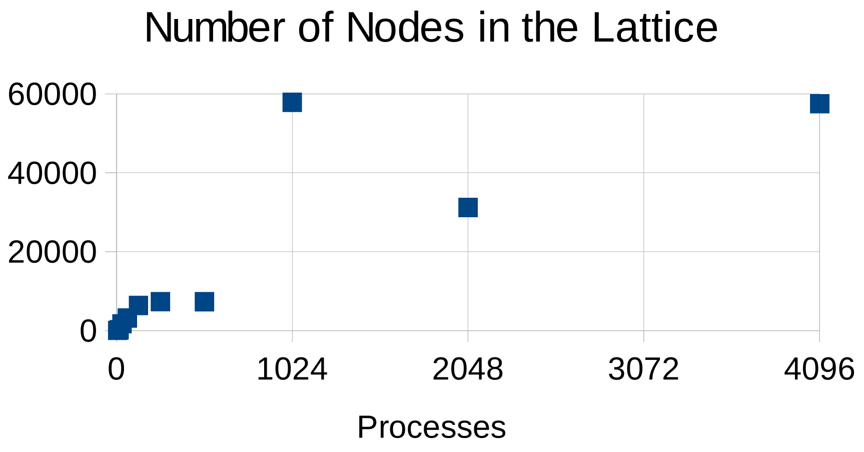

Tracefile Loading Time

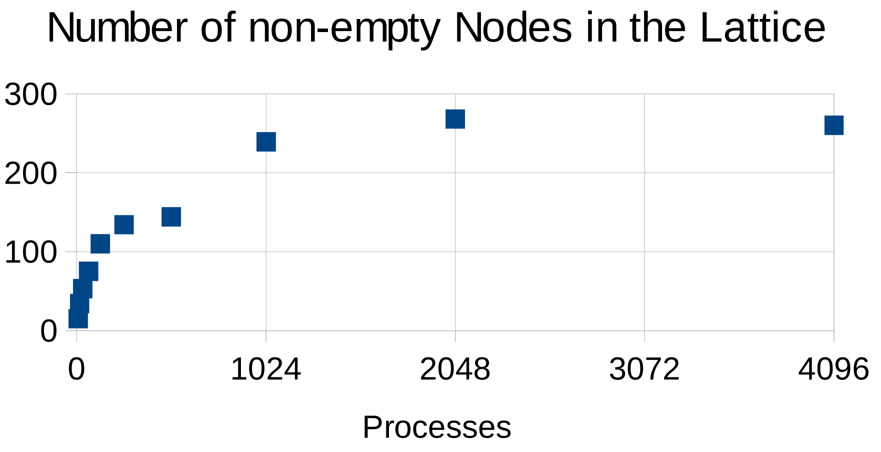

Number of Nodes in the Lattice

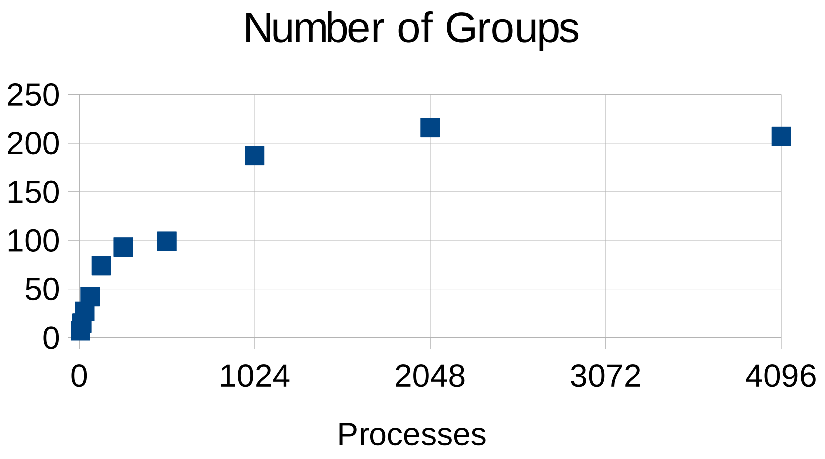

Construction Time div. Loading Time

Classifying Process Traces > Evaluation

- ParaDiS


Number of Nodes in the Lattice

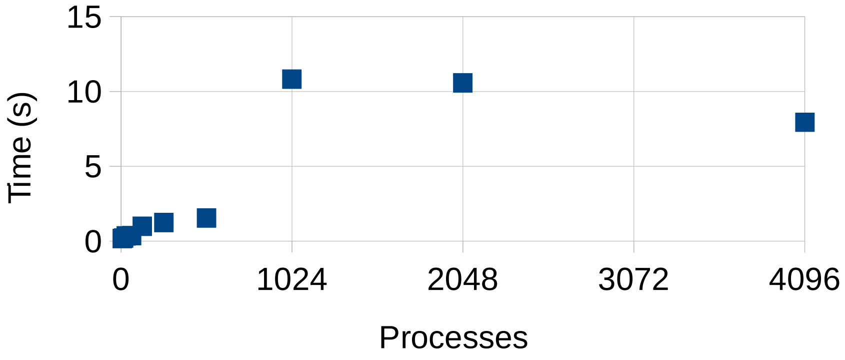

Number of non-empty Nodes in the Lattice


Number of Groups

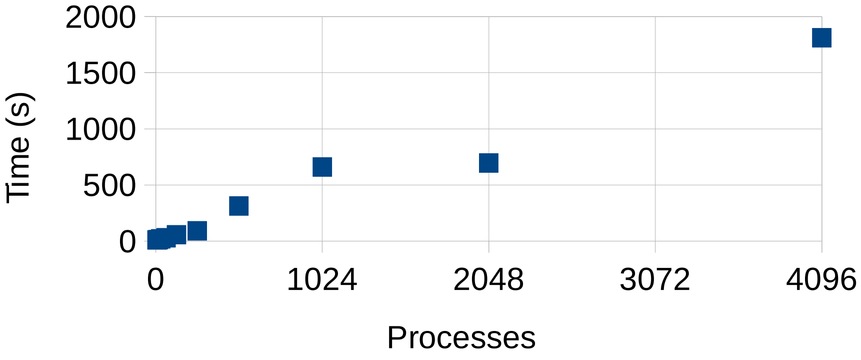
Classifying Process Traces > Evaluation

- ParaDiS, using called functions instead of function pairs

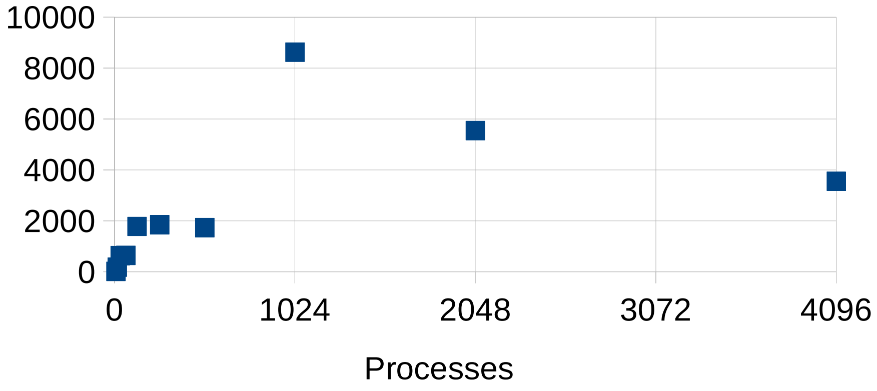
Construction Time



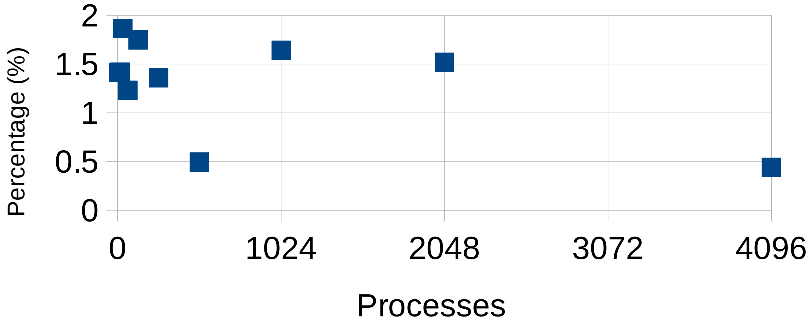
Tracefile Loading Time



Number of Nodes in the Lattice

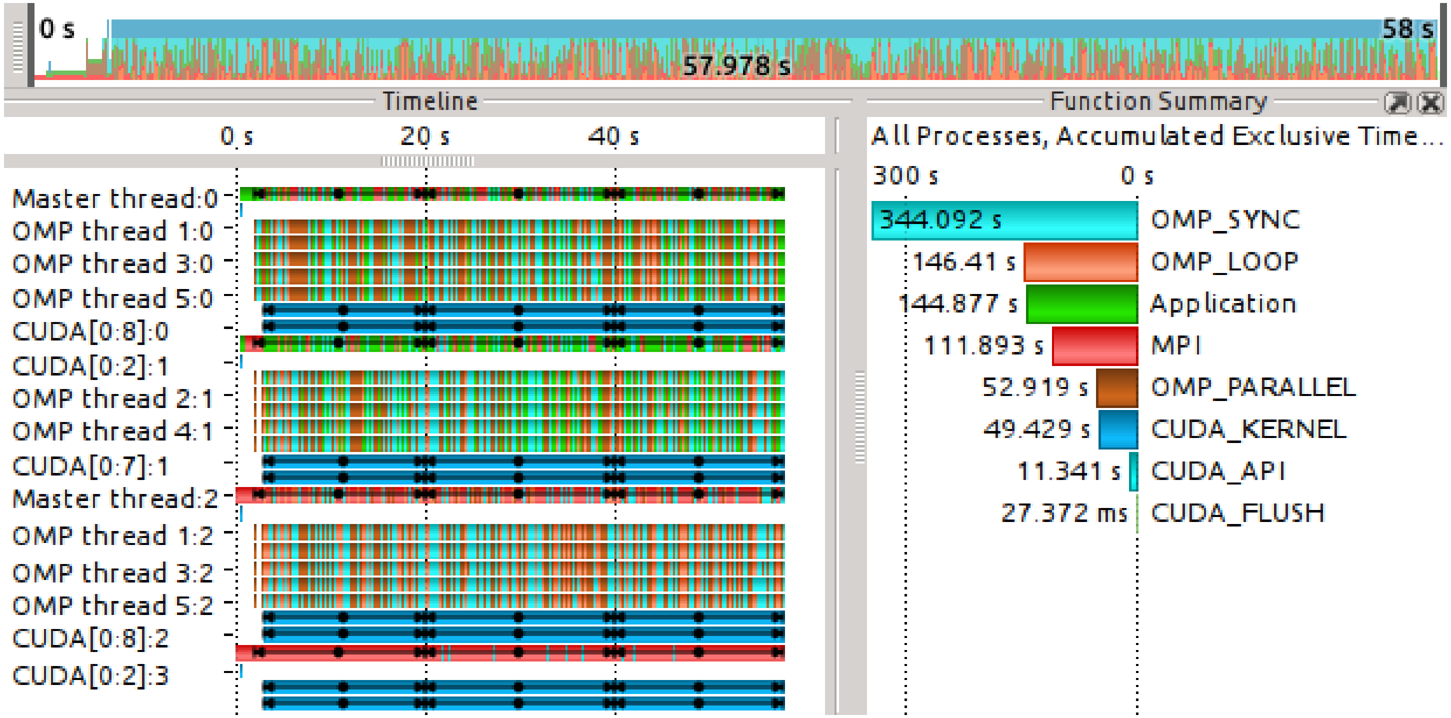


Construction Time div. Loading Time



Classifying Process Traces > Evaluation

- Gromacs
  - Decreasing tracing detail from Masterthread 0 to 3



344.092 s	OMP_SYNC
146.41 s	OMP_LOOP
144.877 s	Application
111.893 s	MPI
52.919 s	OMP_PARALLEL
49.429 s	CUDA_KERNEL
11.341 s	CUDA_API
27.372 ms	CUDA_FLUSH

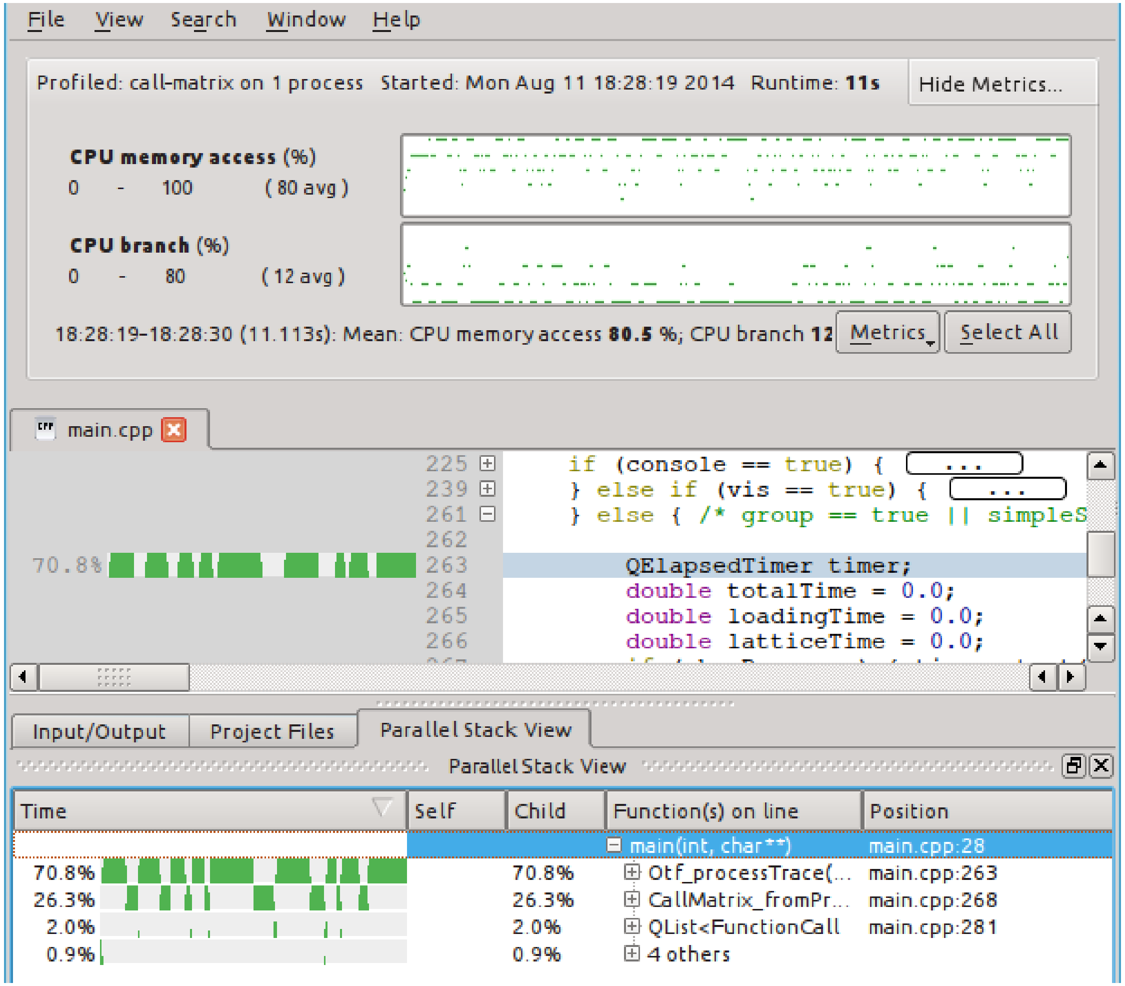
Classifying Process Traces > Evaluation

- Gromacs
  - Subsumption metric matrix, function appearance (not call relations), for all master threads

	0	1	2	3
0		100%	99%	98%
1	76%		99%	98%
2	24%	31%		100%
3	8%	10%	33%	

A Novel Profile Display > State of the Art

- Allinea Map
  - Call path profiling
  - Accumulates over all processes
  - Sliced information (vampir-esque)
  - Various metrics
  - Very intuitive and simple GUI



Profiled: call-matrix on 1 process Started: Mon Aug 11 18:28:19 2014 Runtime: **11s**

CPU memory access (%)
0 - 100 (80 avg)

CPU branch (%)
0 - 80 (12 avg)

18:28:19-18:28:30 (11.113s): Mean: CPU memory access **80.5** %; CPU branch **12**

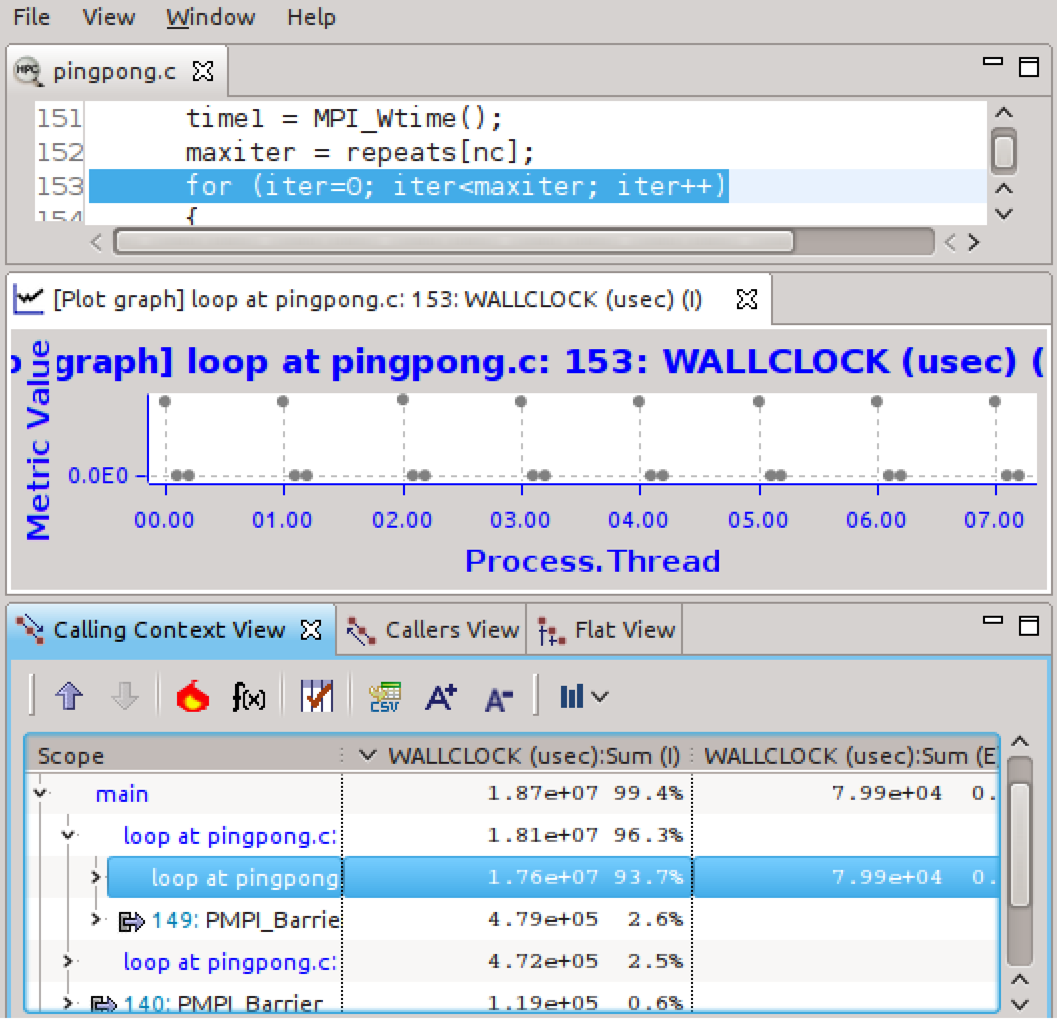
```
  if (console == true) { ... 
  } else if (vis == true) { ... 
  } else { /* group == true || simpleS

70.8%      QElapsedTimer timer;
           double totalTime = 0.0;
           double loadingTime = 0.0;
           double latticeTime = 0.0;
```

Time	Self	Child	Function(s) on line	Position
			⊟ main(int, char**)	main.cpp:28
70.8%		70.8%	⊞ Otf_processTrace(...	main.cpp:263
26.3%		26.3%	⊞ CallMatrix_fromPr...	main.cpp:268
2.0%		2.0%	⊞ QList<FunctionCall	main.cpp:281
0.9%		0.9%	⊞ 4 others	

A Novel Profile Display > State of the Art

- HPCToolkit
  - Call path profiling
  - Accumulates over all processes
  - Plots for one function value x processes
  - Custom metrics

  - Usability is ok

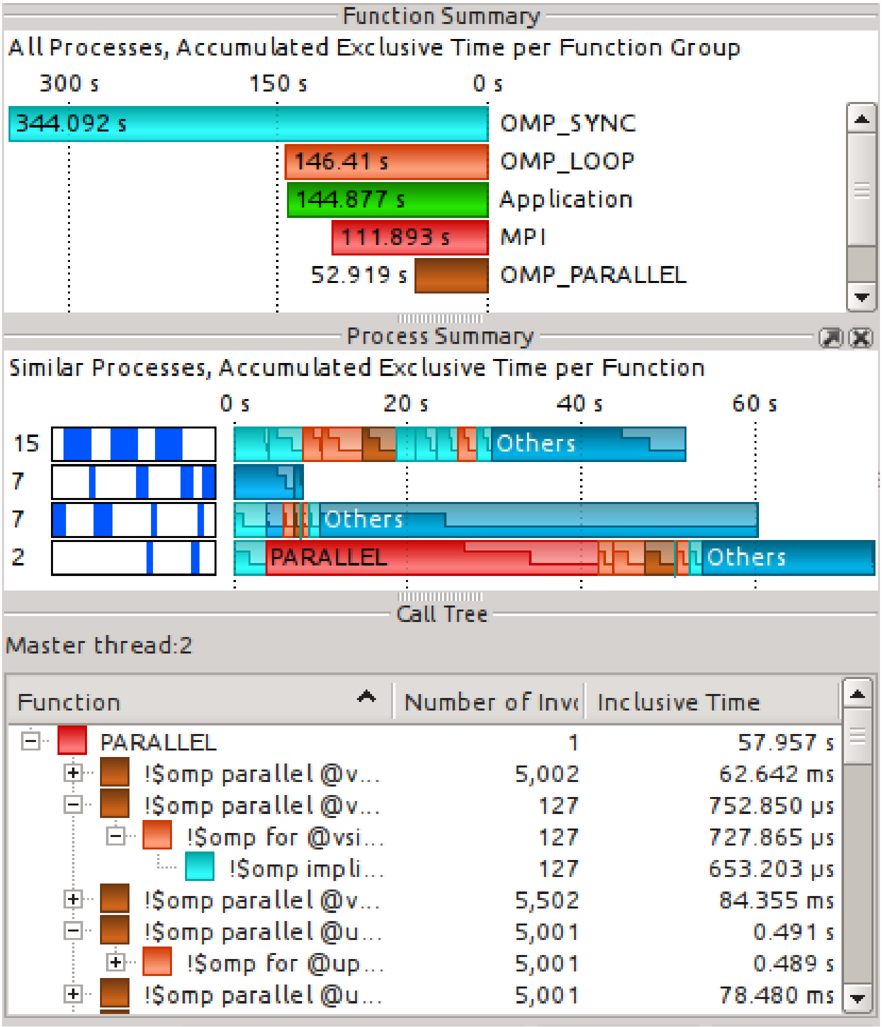


```
        time1 = MPI_Wtime();
        maxiter = repeats[nc];
        for (iter=0; iter<maxiter; iter++)
        {
```

Scope	WALLCLOCK (usec):Sum (I)	WALLCLOCK (usec):Sum (E
main	1.87e+07 99.4%	7.99e+04 0.
loop at pingpong.c:	1.81e+07 96.3%	
loop at pingpong	1.76e+07 93.7%	7.99e+04 0.
149: PMPI_Barrie	4.79e+05 2.6%	
loop at pingpong.c:	4.72e+05 2.5%	
140: PMPI_Barrier	1.19e+05 0.6%	

A Novel Profile Display > State of the Art

- Vampir
  - Powerful profile displays
  - Supports clustering similar profiles
  - Usability is good, but needs practice/knowledge



Master thread:2

Function	Number of Inv(	Inclusive Time
⊟ PARALLEL	1	57.957 s
⊞ !$omp parallel @v...	5,002	62.642 ms
⊟ !$omp parallel @v...	127	752.850 µs
⊟ !$omp for @vsi...	127	727.865 µs
!$omp impli...	127	653.203 µs
⊞ !$omp parallel @v...	5,502	84.355 ms
⊟ !$omp parallel @u...	5,001	0.491 s
⊞ !$omp for @up...	5,001	0.489 s
⊞ !$omp parallel @u...	5,001	78.480 ms

A Novel Profile Display > Outlook

- What can we improve?
  - Replace the Function Summary, Call Tree and Process Summary displays in Vampir with one unified display.
  - Usability
  - Comparative analysis

A Novel Profile Display > Outlook

- Ideas: General Usability
  - Fewer bars, maybe none
  - Improved search functionality
  - Proper undo stack
  - Context menu that is actually context-sensitive
  - ...

A Novel Profile Display > Outlook

- Ideas:
  - Show useful and simple information at first sight
  - Enable easy regrouping of functions
  - Highlighting points of interest
    - Special sorting options
    - ?
  - Display timing variations between different calls or processes using e.g. box plots
  - Comparative Analysis:
    - Filter processes using the new clustering
    - ?

Conclusion

- Introduced a structural similarity metric
- Developed methods to cluster and compare processes in $< O(n^2)$ steps
  - works with any set-based similarity metric
- First steps towards a better profile viewer

Future Work

- Further scalability testing of the clustering
- Develop a novel, scalable profile viewer
- Use the developed methods in an actual visualisation

References

- [1] 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.
- [2] Matthias Weber, Kathryn Mohror, Martin Schulz, Bronis R. de Supinski, Holger Brunst, and Wolfgang E. Nagel. Alignment-Based Metrics for Trace Comparison. In *Euro-Par 2013 Parallel Processing*, pages 29–40. Springer, 2013.
- [3] 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.

References

- [4] Robert D Falgout and Ulrike Meier Yang. hypre: A library of high performance preconditioners. In Computational Science—ICCS 2002, pages 632–641. Springer, 2002.
- [5] ParaDiS. http://paradis.stanford.edu, 2014-08-27.
- [6] Allinea Map. http://www.allinea.com/products/map, 2014-08-27.
- [7] HPCToolkit. http://hpctoolkit.org, 2014-08-27.

Thank You!