An Accurate Tool for Modeling, Fingerprinting, Comparison, and Clustering of Parallel Applications Based on Performance Counters

V. Ramos, C. Valderrama, P. Manneback, and **S. Xavier-de-Souza**

IPDPS 2019 Rio de Janeiro

UMONS Université de Mons

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

CONTEXT

- **Hardware Performance Counters** (HPC) are special registers available on most modern processors
- HPCs are capable of counting **hundreds of micro-architectural events** such as instructions executed, cache-hit, branches miss-predicted, energy estimation and much more.
- Exploiting this HPCs requires an **intimate knowledge of the micro-architecture** and kernel API, as well as an awareness of an ever increasing complexity.
- **Still lacking of high-level APIs.**

RELATED WORK

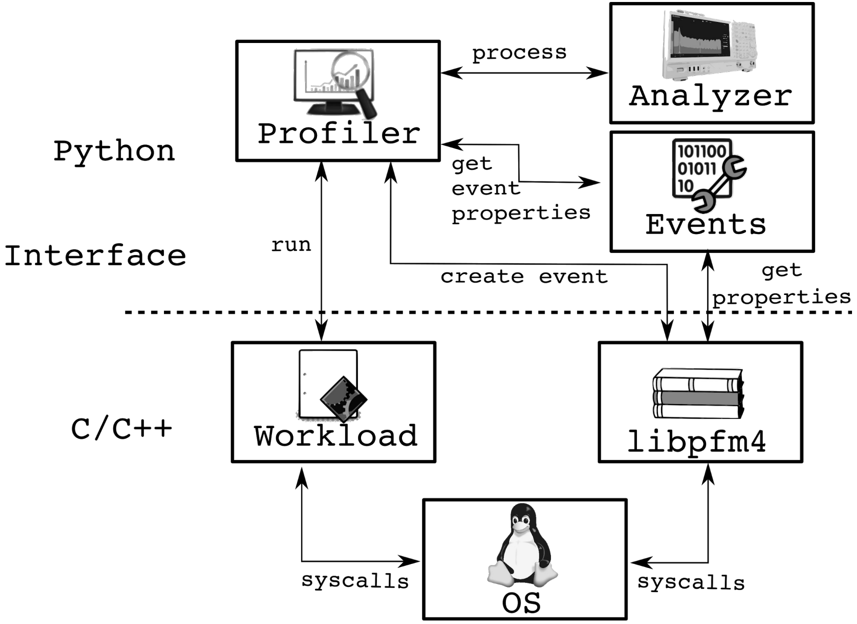
- PAPI developed in C
- A few non-official PAPI libraries ported to Python.
  - Python version has a considerable overhead
  - Does not show an easy way to
    - create raw events or
    - control low-level events
- Intel VTUNE, Perfctr and Perfmon2
  - Need special drivers
- **Linux came up with a performance counters subsystem**
  - a complete set of configurations for hardware and software events

READING PERFORMANCE COUNTERS

- The configuration of the counters is done via **Model-Specific Registers** (MSR)
- Operating systems provide an abstraction of these hardware capabilities to **access counters and MSRs**.
- On Linux accessible via special file descriptors opened via the **perf_event_open() system call**.
- Counters can be **read with different forms**:
  - Polling (when an event happen)
  - Interruption
  - **Time**

PYTHON TOOL TO COLLECT HARDWARE PERFORMANCE COUNTERS

- Developed with Python and C++
- *Profiler*
  - Python API for **accessing, configuring and analyze** performance counters
- *Events*
  - **high level api** for finding available events in the system
- *Workload*
  - execute and sample the counters
- *libpfm4*
  - **helper library** to find and create performance events
- *Analyzer*
  - responsible for the **post-processing**, filtering and interpolation



READING PERFORMANCE COUNTERS

Installation on ubuntu:

```
sudo apt install python-dev swig libpfm4-dev
pip install performance-features
```

Creating 3 event groups and sampling over time:

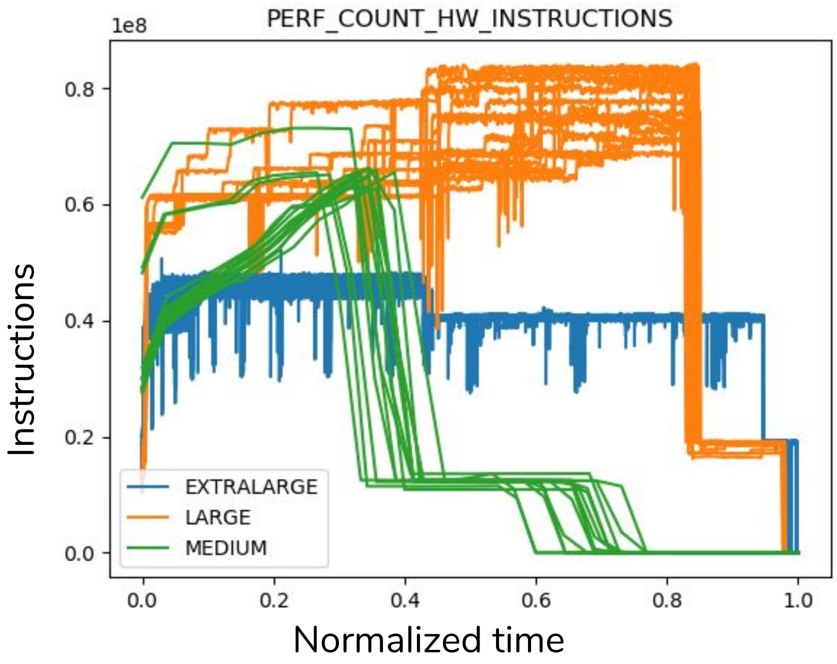
```
from profiler import *
try:
    events= [['PERF_COUNT_HW_INSTRUCTIONS'],
             ['PERF_COUNT_HW_BRANCH_INSTRUCTIONS','PERF_COUNT_HW_BRANCH_MISSES'],
             ['PERF_COUNT_SW_PAGE_FAULTS']]
    perf= Profiler(program_args= ['/bin/ls','/'], events_groups=events)
    data= perf.run(sample_period= 0.01)
    print(data)
except RuntimeError as e:
    print(e.args[0])
```

ACCURACY COMPARISON

	Average*10^6					Standard Deviation			
Counters	Pined values	Linux API	PAPI	PAPI Python	Our tool	Linux API	PAPI	PAPI Python	Our tool
INSTRUCTIONS_RETIRED	226.99	227	227	225.9	227	396	133	337763	175
BRANCH_INSTRUCTIONS_RETIRED	9.24	9.25	9.25	9.24	9.25	297	208	8485	91
BR_INST_RETIRED:CONDITIONAL	8.22	8.22	8.22	8.21	8.22	0	0	3383	0
MEM_UOP_RETIRED:ANY_LOADS		2484.18			2484.16	37399			38953
MEM_UOP_RETIRED:ANY_STORES		189.96			189.96	1513			687
UOPS_RETIRED:ANY		12291.08			12290.9	345246			333298
PARTIAL_RAT_STALLS:MUL_SINGLE_UOP		0.6			0.6	1222			521
ARITH:FPU_DIV		5.8			5.8	1760			1544
FP_COMP_OPS_EXE:X87		48.79			48.79	1283			3311
INST_RETIRED:X87		17.2			17.2	4			3
FP_COMP_OPS_EXE:SSE_SCALAR_DOUBLE		5.4			5.4	1547			2097

POST-PROCESSING

- Ideal hardware performance counters provide exact deterministic results…
- Some HPCs are non-deterministic even in controlled environments, others present overcounting and some are just wrong
- Need for post-processing



POST-PROCESSING

1 **HPC gathering**

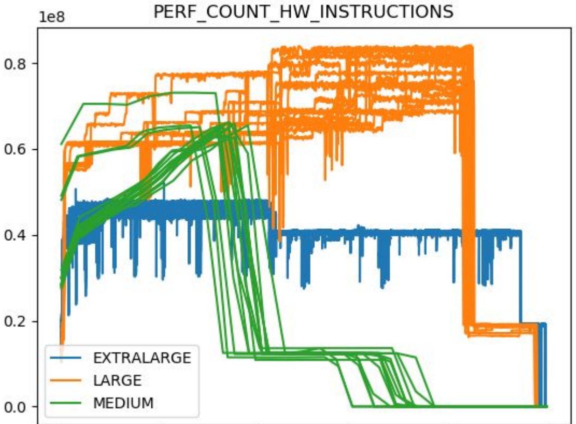
Collect HPC on multiple runs

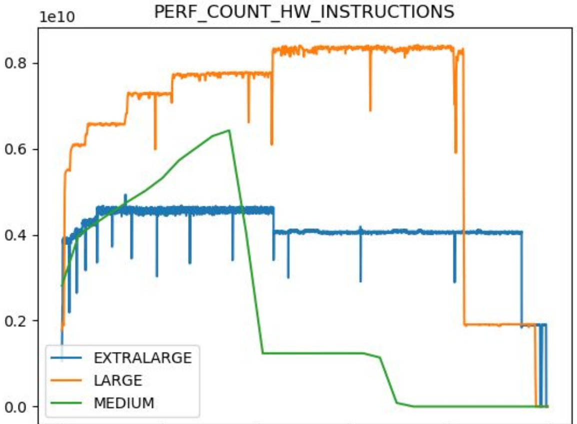
2 **Single curve**

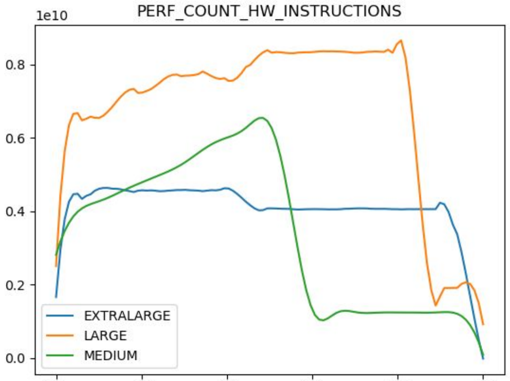
Remove outliers using median filter and calculate the mean curve

3 **Smooth the curve**

Interpolate the curve and filter to get a smoother curve

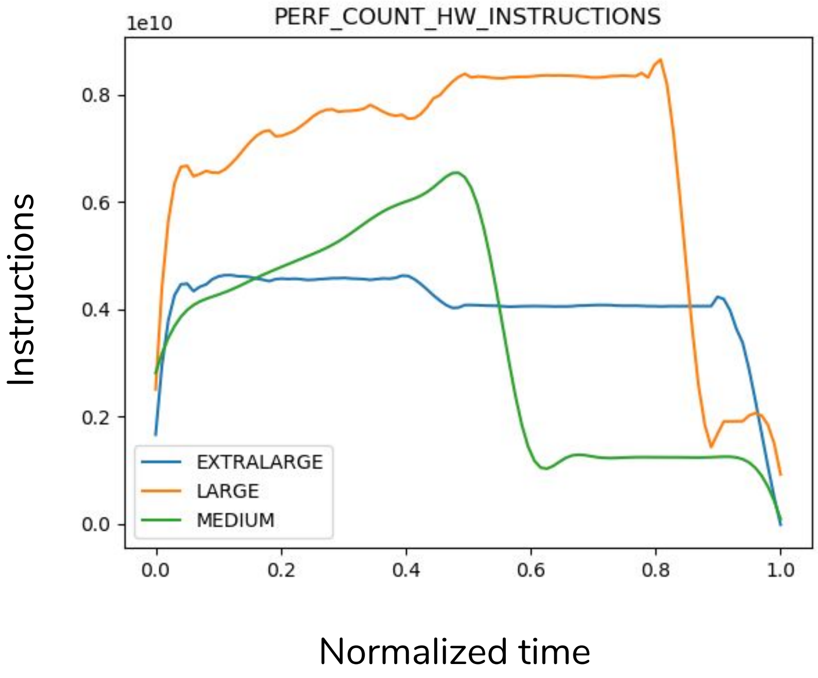






Why Post-processing for clustering ?

- Removing outliers improve the classification
- Smothering focus the classification on the shape of the curve
- Interpolation and execution time normalization makes the curve have the same number of points



CLUSTERING

- 30 applications of the Polybench with 3 different inputs
- Metric input size defined as the ratio of instructions by memory instructions

$$I_{sz} = \frac{I}{I_m}$$

- Using the canberra distance to measure distance between curves

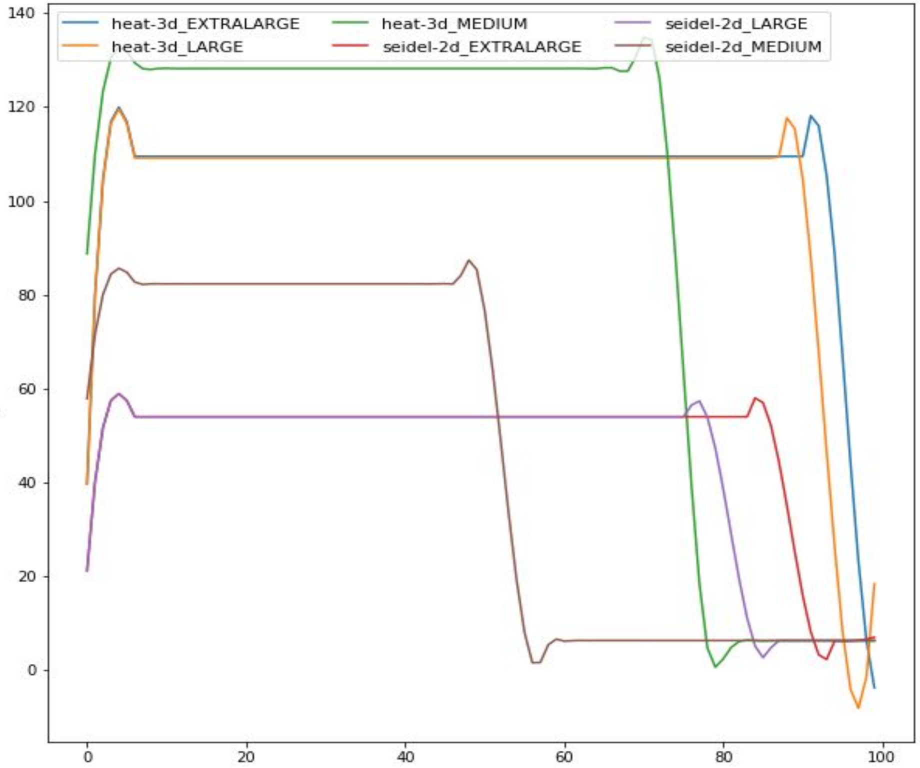
$$d(p,q) = \sum_{i=1}^{n} \frac{|p_i - q_i|}{|p_i| + |q_i|}$$

- Clustering method:
  - Linkage method of Ward

CLUSTERING

Cluster 1

$$\frac{I}{I_m}$$

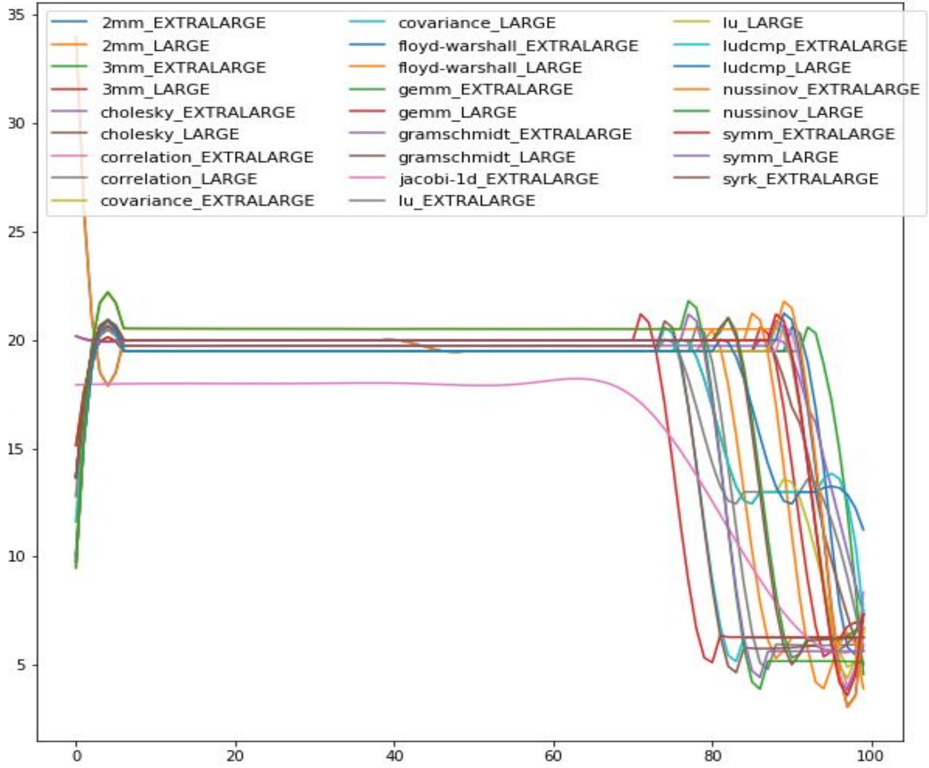


Normalized time

CLUSTERING

Cluster 2

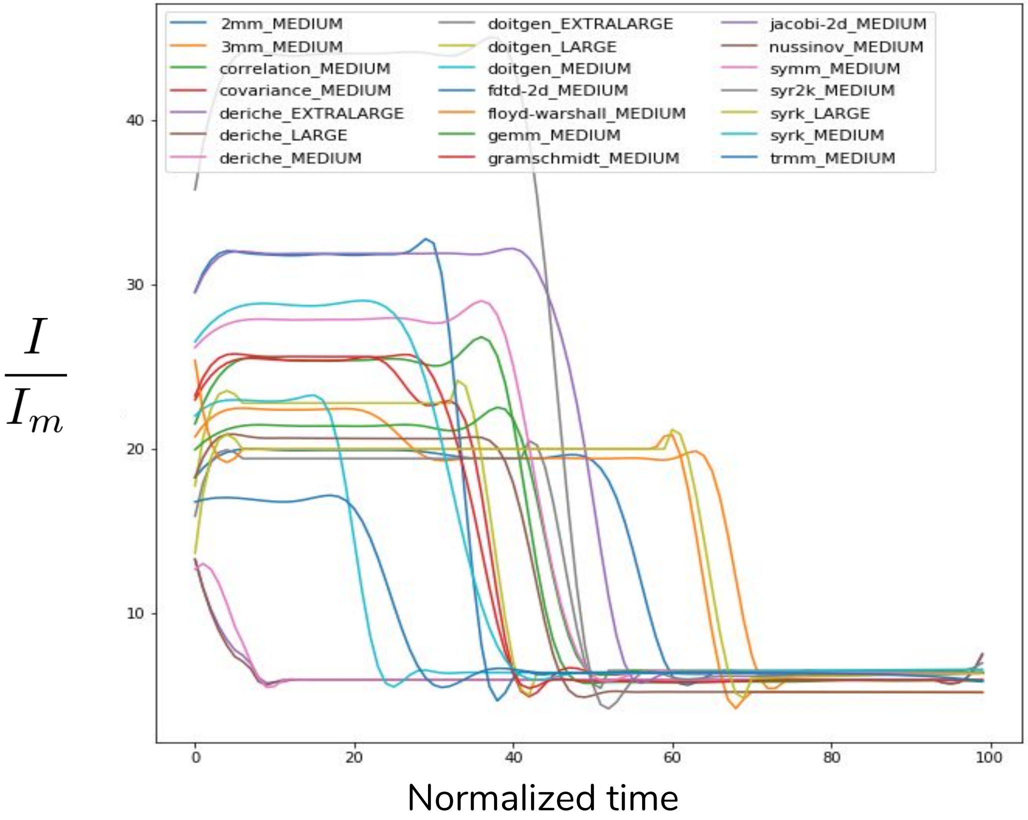
$$\frac{I}{I_m}$$



Normalized time

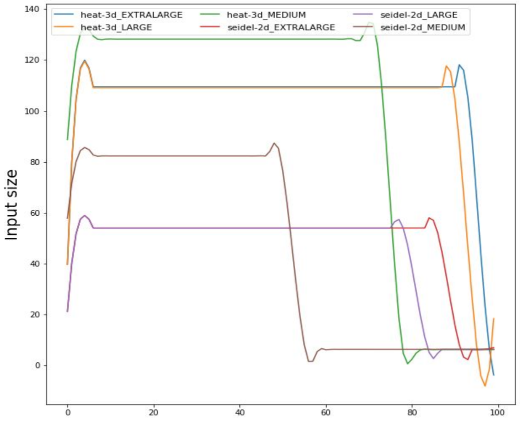
CLUSTERING

Cluster 3

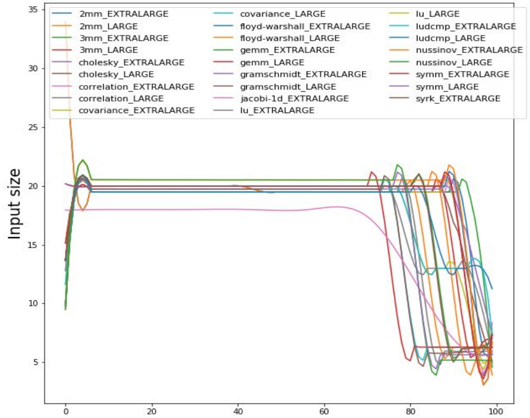


CLUSTERING

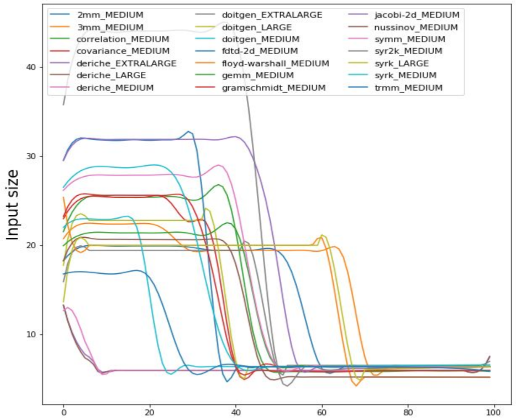
Cluster 1



Cluster 2

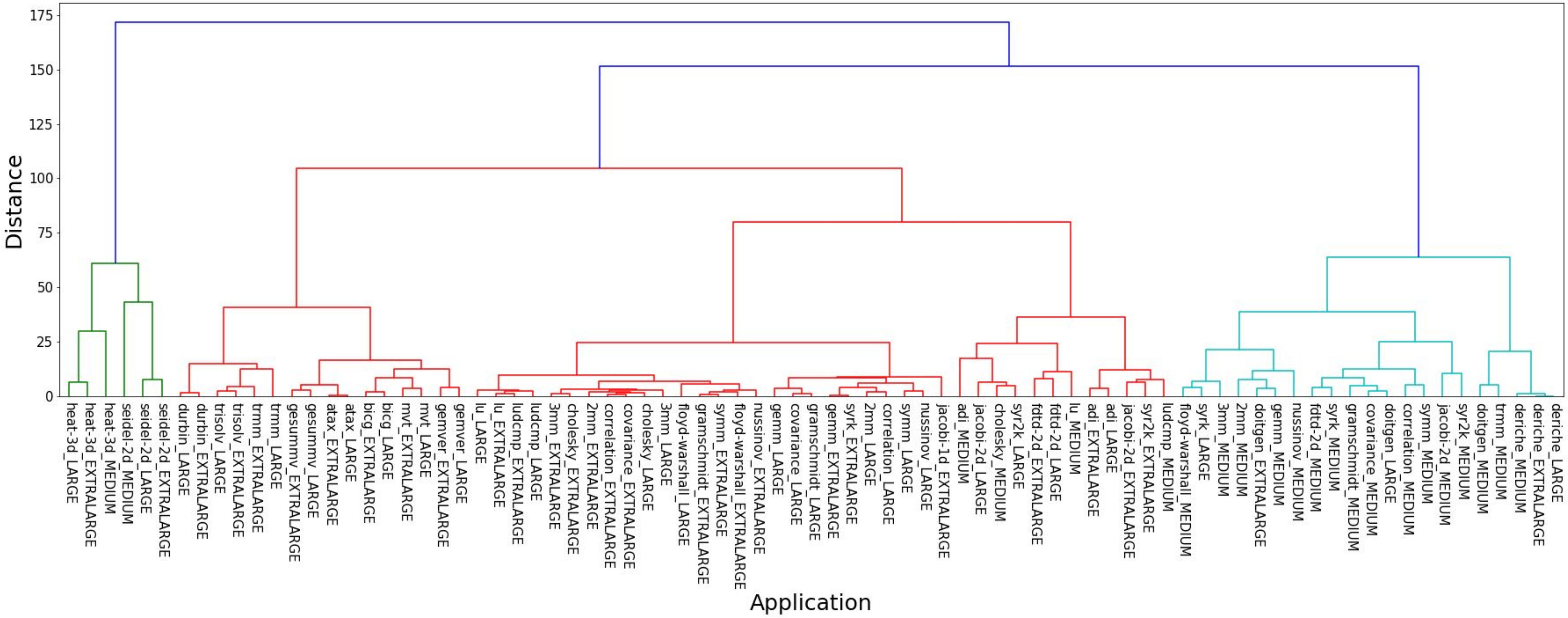


Cluster 3



- Curves that have similar shape have also been classified as the same clusters
  - Regardless of scale on vertical and horizontal axis
- 24 applications with different inputs were classified in the same cluster

CLUSTERING



- Dendrogram shows how close applications are from each other

UMONS Université de Mons

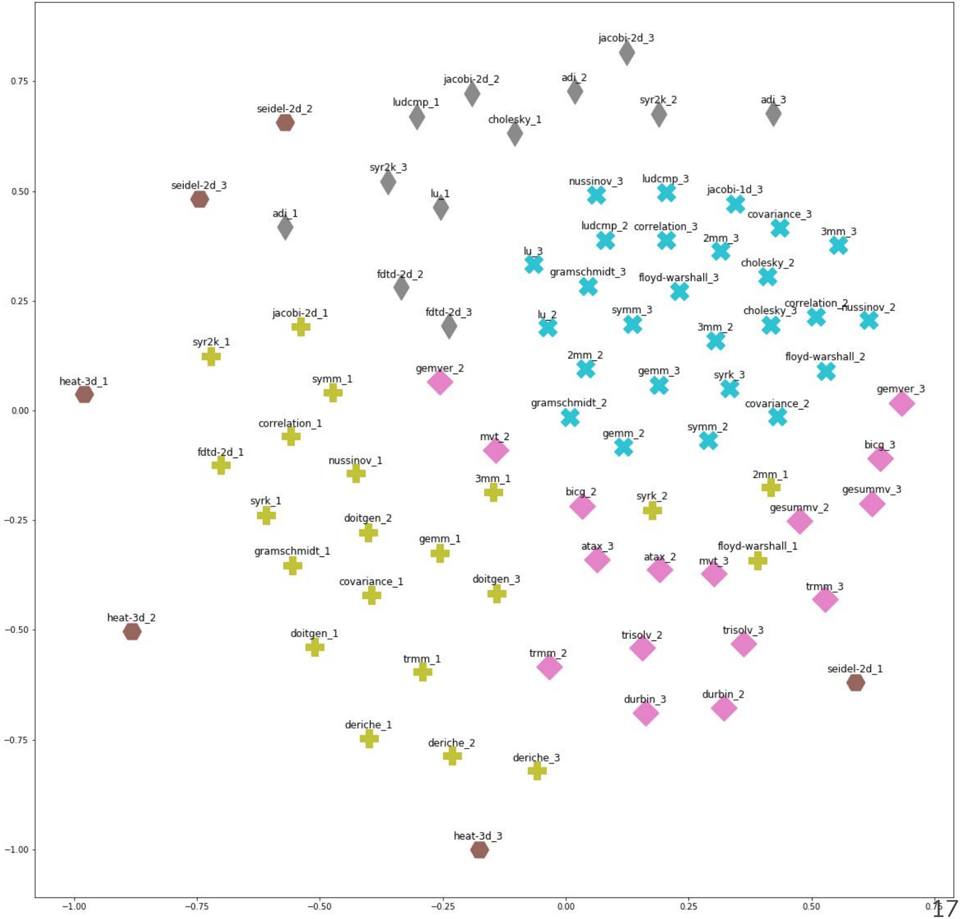
UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

CLUSTERING

IPDPS 2019 Rio de Janeiro

Clusters

- 2mm, 3mm, cholesky, correlation, covariance, floyd-warshall, gemm, gramschmidt, lu, ludcmp, nussinov, symm
- deriche, doitgen, syrk
- adi, fdtd-2d, jacobi-2d, syr2k
- atax, bicg, durbin, gemver, gesummv, mvt, trisolv, trmm
- heat-3d, seidel-2d



CONCLUSIONS

- Our tool exposes linux API to Python
- Overhead similar or lower the established APIs
- High abstraction and simplified configuration
- Fingerprint programs and compute similarities between programs given a metric
- Clustering reduces application space

FUTURE WORK

- Create a data set of applications behavior for automatic classification of programs.
- The idea is to have a set of clusters that can describe most applications in this way we can know specific behaviors of the applications.
- Can be applied to various areas where application classification is needed, for example
  - Benchmark creation,
  - Dynamic Frequency and Voltage Scaling