Parallel Programming Tools in Python

Guillermo Cornejo-Suárez
Esteban Meneses

Escuela de Computación, Tecnológico de Costa Rica
Colaboratorio Nacional de Computación Avanzada, Centro Nacional de Alta Tecnología

August 23rd, 2017

Why Python?

- Five rankings on the popularity of programming languages, with different methodologies, list Python among the first 4.
- In 2015 $\sim 10\%$ of all computer jobs at TACC were Python applications.
- Strong scientific code base.
- Ease of use.

Why Parallel Python?

- Multi-core and many-core processors represent industry's answer to Moore's technical halt.
- As complexity grows, applications must take advantage of modern architectures.
- New applications must be parallel.

Parallel Python as research field

- Popularity
- Growth potential
- Absence of *de facto* standard for parallel computing.

Providing parallelism in the Python programming language is a promising research field.

Methodology

Search engines with search key “parallel python”

- Indexed publications:
  - IEEExplore
  - ACM Digital Library
  - ScienceDirect
  - SpringerLink
- Google Scholar
- Commercial Google

Results were sorted by date in descending order from 2016 until 1993.

Inclusion criteria

1. The project provides some kind of parallelism to the Python programming language.
2. The project aims general programming rather than domain-specific.

35 projects were included

Classification criterion

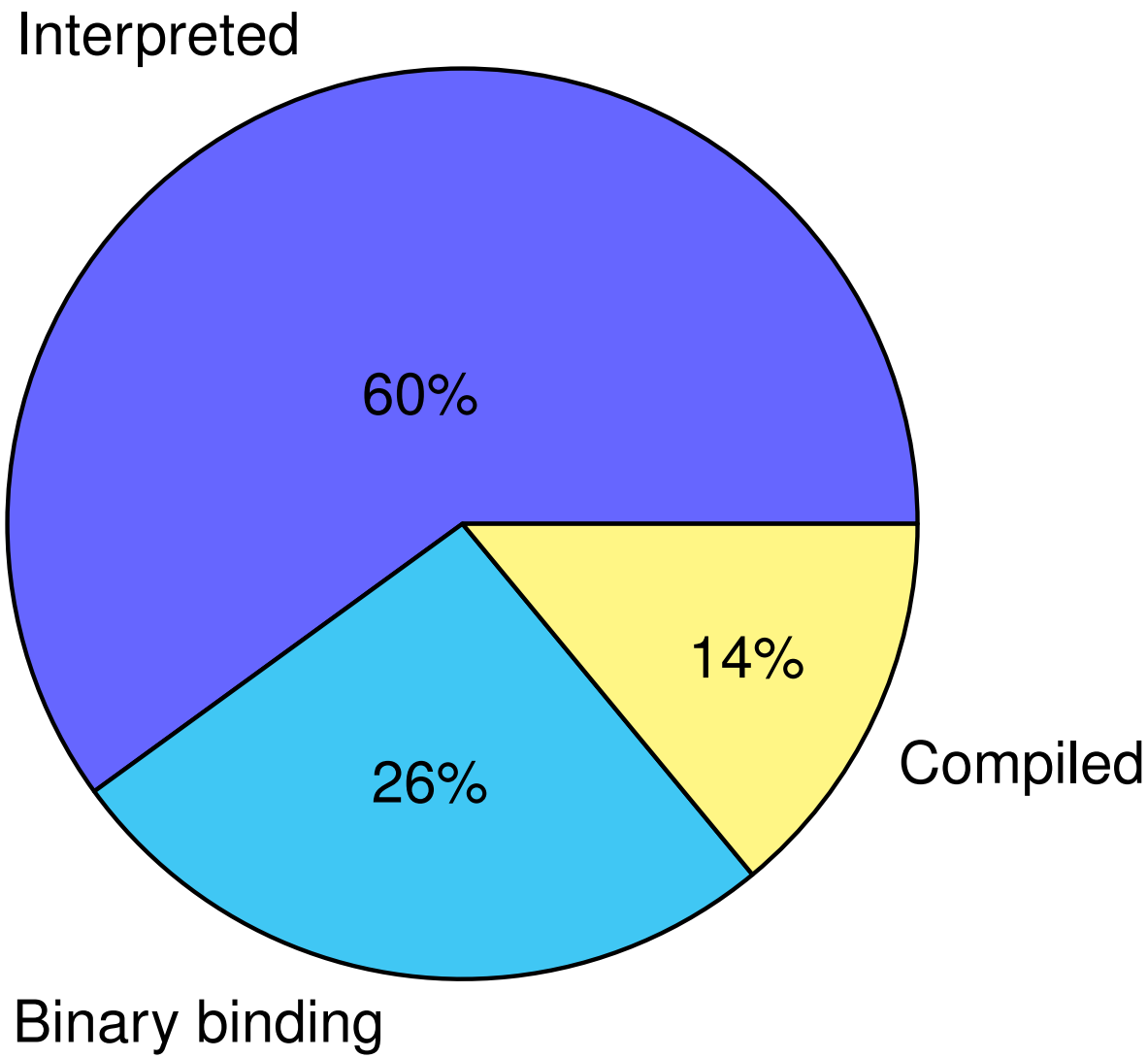
1. Execution strategy: Interpreted, binary binding, compiled.
2. Parallel paradigm: Data, Task, Message Passing, MapReduce.
3. Vector Data oriented.
4. Language support: Full, Subset.
5. Code modifications: Function calls, Job submission, Annotations, None.
6. Target platform: SMP, Clusters, GPU.

Classification table

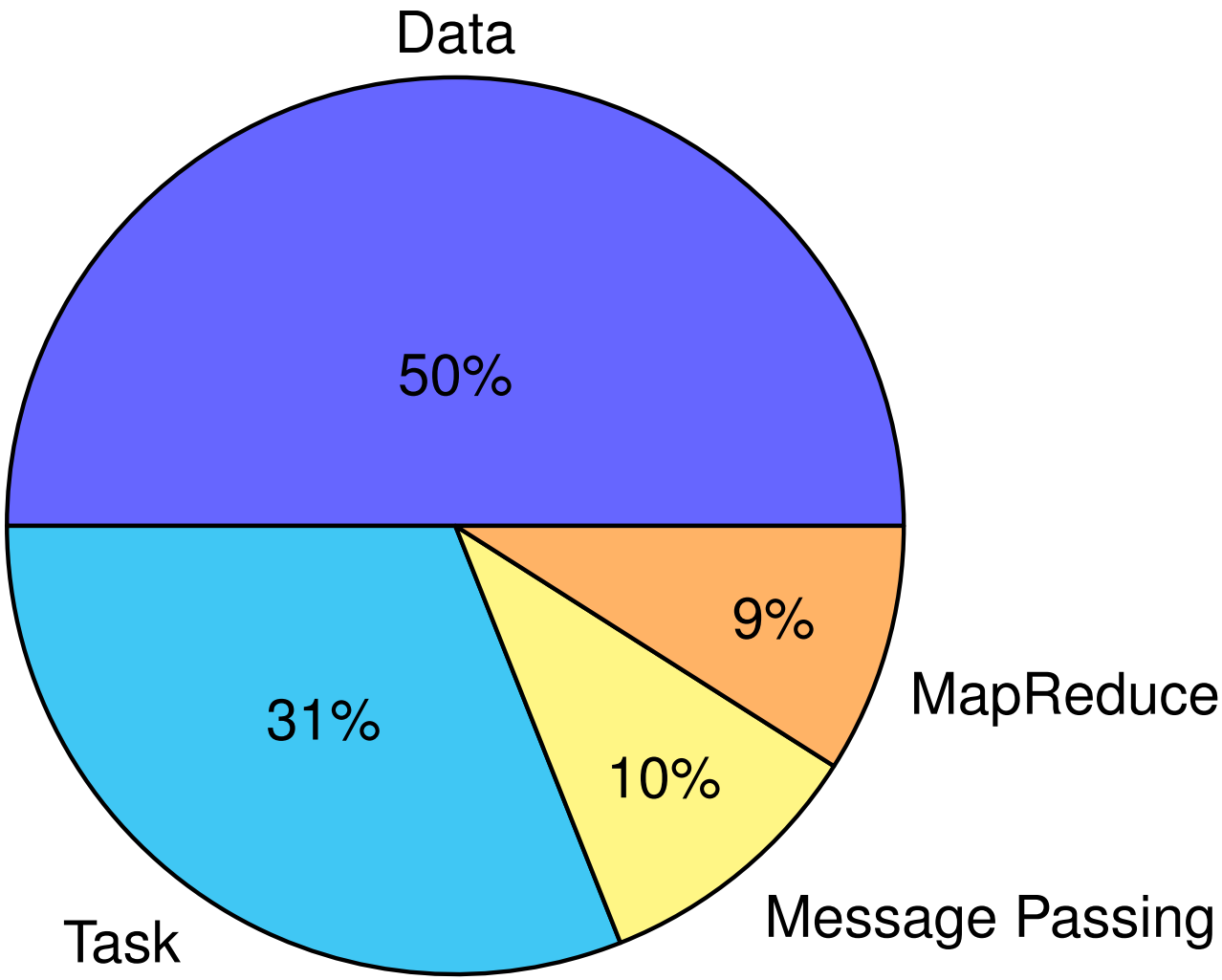
TABLE I
CLASSIFICATION OF TOOLS THAT PROVIDE PARALLELISM IN THE PYTHON PROGRAMMING LANGUAGE.

Project	Execution strategy	Parallel paradigm	Vector data oriented	Language support	Code modifications	Parallel platform	Latest release
Bohrium [14]	Interpreted	Data	Yes	Full, Python 2	None	SMP, GPU, Clusters	0.3, Apr-2016
PyStream [15]	Compiled	Data	Yes	Subset, Python 2	None	GPU	0.1, Jul-2011
Dask.array [16]	Interpreted	Data	Yes	Full, Python 3	FunCall	SMP, Clusters	0.13.0, Jan-2017
PupyMPI [17]	Interpreted	MsgPsg	No	Full, Python 2	FunCall	SMP, Clusters	0.9.5, May-2011
Papy [18]	Interpreted	Task	No	Full, Python 2	JobSub	SMP, Clusters	1.0.8, Nov-2014
GAiN [19]	Binary binding	Data	Yes	Full, Python 2	FunCall	Clusters	1.0, 2009
Global Arrays [20]	Binary binding	Data	Yes	Full, Python 2	FunCall	Clusters	5.5, Aug-2016.
mpi4Py [21]	Binary binding	MsgPsg	No	Full, Python 2-3	FunCall	SMP, Clusters	2.0.0, Oct-2015
Pythran [22]	Compiled	Data	Yes	Subset, Python 3	Annotations	SMP	0.7.6.1, Jul-2016
ASP [23]	Binary binding	Data, Task	No	Full, Python 2	JobSub	SMP, GPU	0.1.3.1, Oct-2013
Dispel4py [24]	Interpreted	Data, Task	No	Full, Python 2-3	JobSub	SMP, Clusters	1.2, Jun-2015
PMI [25]	Interpreted	Data	No	Full, Python 2-3	FunCall	SMP, Clusters	1.0, Dec-2009
Jit4OpenCL [26]	Compiled	Data	Yes	Full, Python 2	Annotations	SMP, GPU	1.0, 2010
MRS [27]	Interpreted	MapRed	No	Full, Python 2-3	FunCall	Clusters	0.9, Nov-2012
Pydron [28]	Interpreted	Task	No	Subset	Annotations	Clusters	–
CoArray [29]	Interpreted	Data	Yes	Full, Python 2	FunCall	Clusters	2004
PyCuda, PyOpenCL [30]	Binary binding	Data	Yes	Full, Python 2-3	FunCall	SMP, GPU	2016.2, Oct-2016
SCOOP [31]	Interpreted	Task	No	Full, Python 2-3	JobSub	SMP, Clusters	0.7.1.1, Ago-2015
DistArray [32]	Interpreted	Data	Yes	Full, Python 2-3	JobSub	SMP, Clusters	0.6, Oct-2015
Dispy [33]	Interpreted	Data, MapRed	No	Full, Python 2-3	JobSub	SMP, Clusters	4.6.17, Sep-2016
IpyParallel [34]	Interpreted	Data, Task	No	Full, Python 2-3	JobSub	SMP, Clusters	5.3.0, Oct-2016
PyRo [35]	Interpreted	MsgPsg	No	Full, Python 2-3	Annotations, FunCall	Clusters	4.50, Nov-2016
Parallel python [36]	Interpreted	Task	No	Full, Python 2-3	JobSub	SMP, Clusters	1.6.5, Jul-2016
JUG [37]	Interpreted	Task	No	Full, Python 2-3	Annotations, FunCall	SMP, Clusters	1.3.0, Nov-2016
Multiprocessing [38]	Interpreted	Task, Data	No	Full, Python 2-3	FunCall	SMP, Clusters	3.6, Jul-2016
Copperhead [39]	Binary binding	Data	Yes	Subset, Python 2	Annotations	GPU	2013
Celery [40]	Interpreted	Task	No	Full, Python 2-3	Annotations, FunCall	SMP, Clusters	4.0.0, Nov-2016
Disco [41]	Interpreted	MapRed	No	Full, Python 2	Annotations, FunCall	SMP, Clusters	0.5.4, Oct-2014
Spark [42]	Binary binding	Task	No	Full, Python 2-3	FunCall	Clusters	2.0.2, Nov-2016
Theano [43]	Binary binding	Data	Yes	Full, Python 2-3	FunCall	SMP, GPU, Clusters	0.8.2, Apr-2016
Numba [44]	Compiled	Data	Yes	Full, Python 2-3	Annotations	SMP, GPU	0.29.0, Oct-2016
Joblib [45]	Interpreted	Task	No	Full, Python 2-3	JobSub, Annotations	SMP	0.10.3, Oct-2016
Hadoopy [46]	Binary binding	MapRed	No	Full, Python 2	JobSub	Clusters	0.5.0, Jun-2012
PyMW [47]	Interpreted	Task	No	Full, Python 2	FunCall	Clusters	0.4, Jun-2010
Pyfora [48]	Compiled	Data	No	Subset, Python 2	None	Clusters, SMP	0.5.8, Set-2016

Remarks: Execution strategy



Remarks: Parallel paradigm



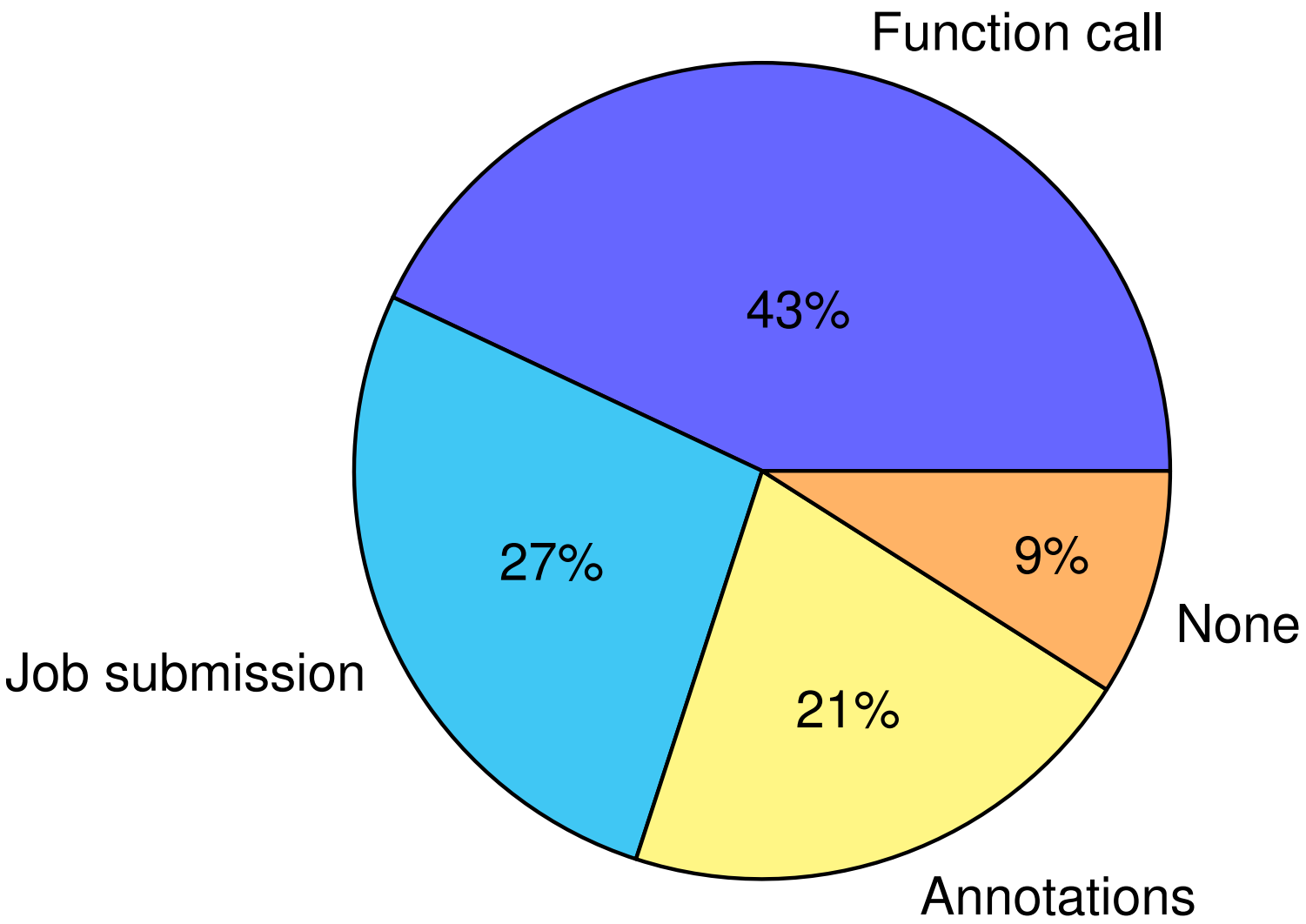
Remarks: Vector data oriented

- 37% of the projects provide vectors as principal structure.
- All of the expose data parallelism.

Remarks: Language support

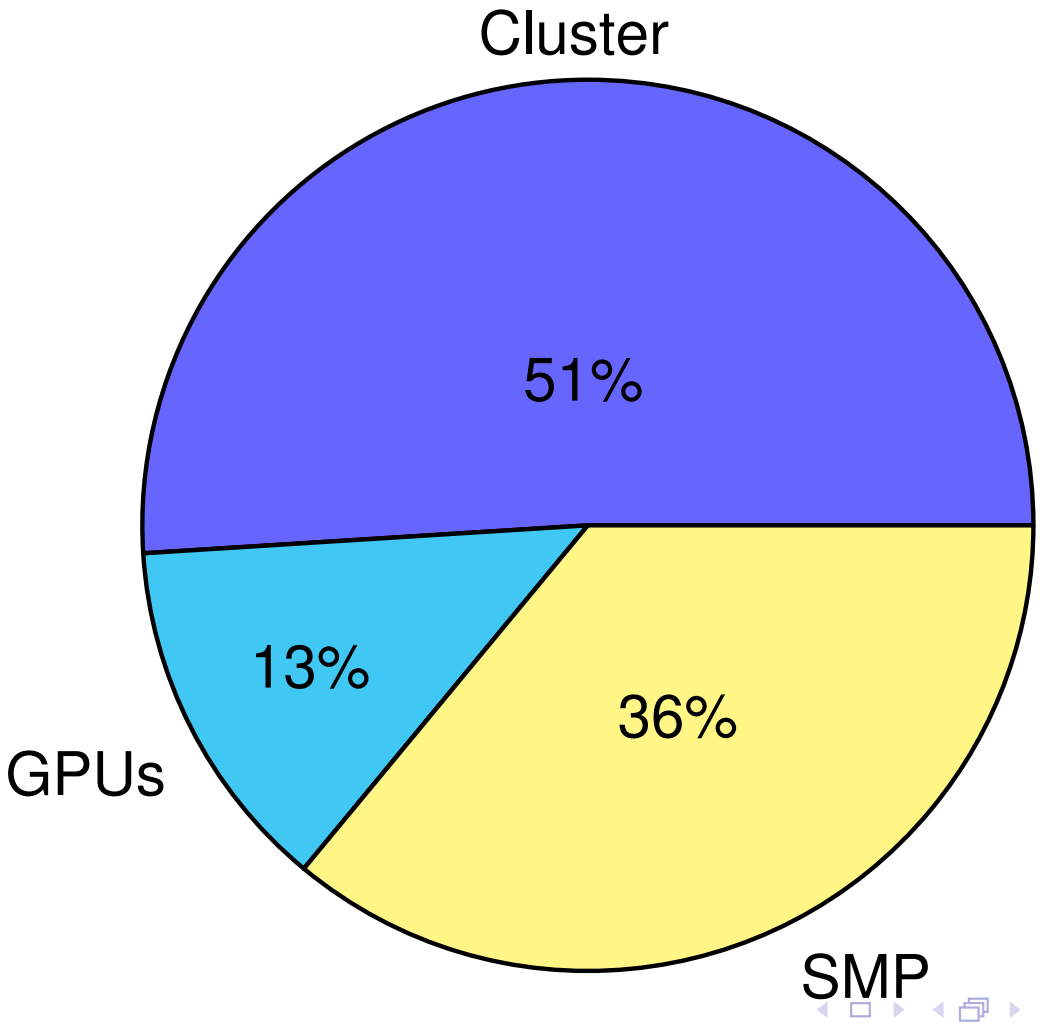
- 86% of the projects supports the full language.
- 40% only support Python 2.
- Plans to upgrade to Python 3 are mixed, strongly depending on project continuity.

Remarks: Code modifications



Remarks: Parallel platform

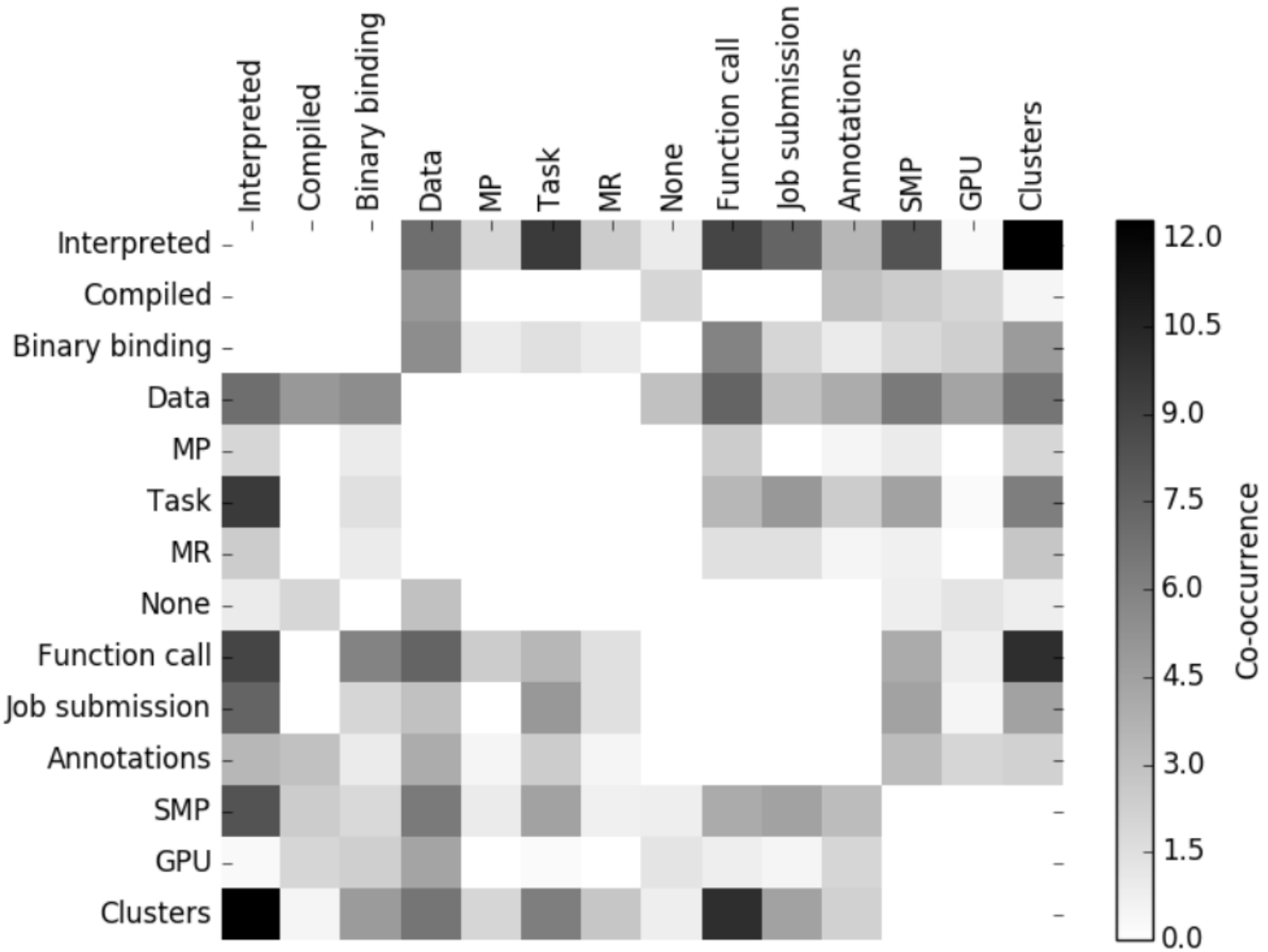
This classification is no mutually exclusive, many projects run on two or three platforms, percentages are then pondered accordingly.



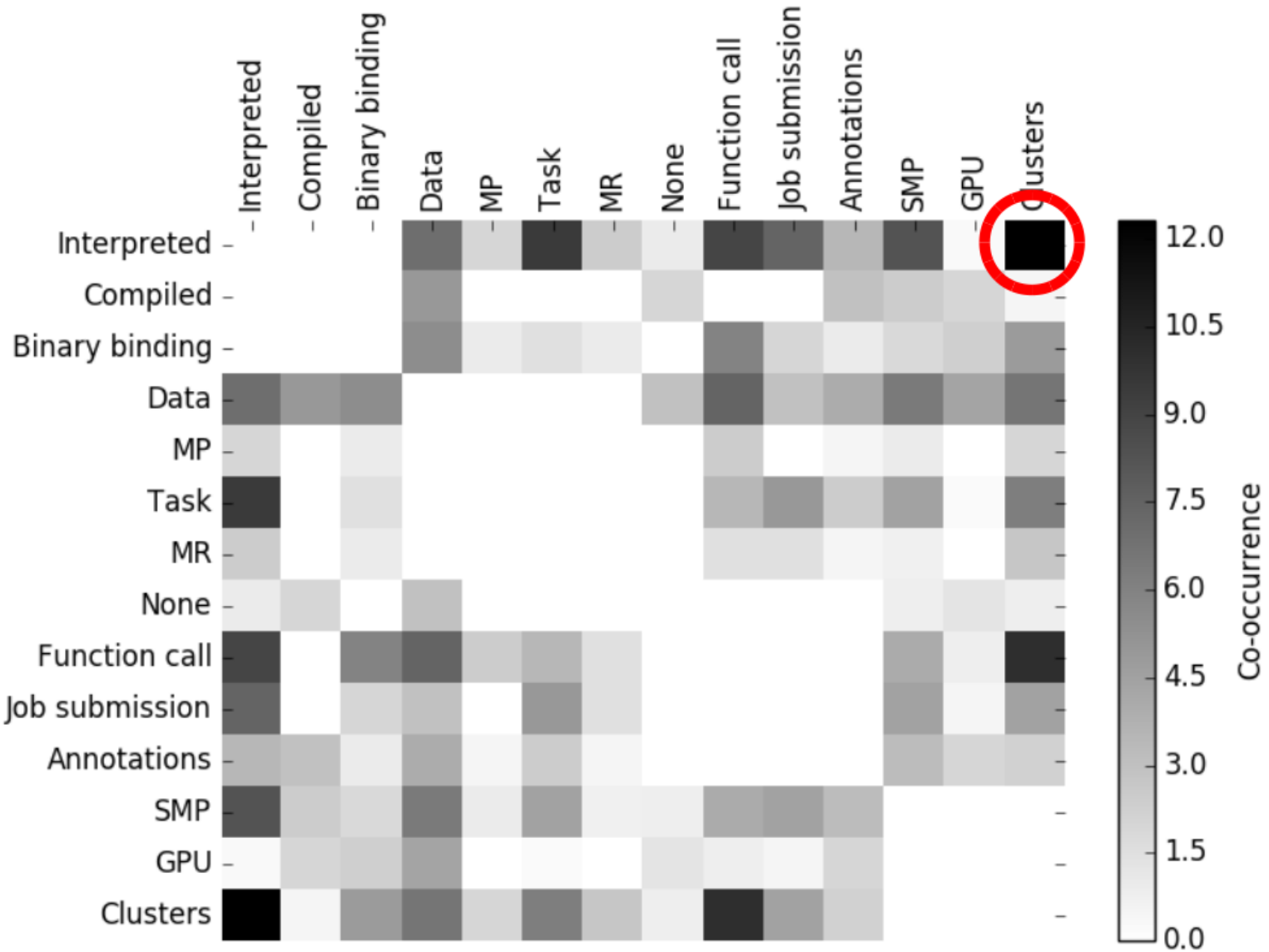
Remarks: Latest release

- 48% of the projects released a new version the last year.
- 60% released a new version in the past two years.
- 46% have an stable release.

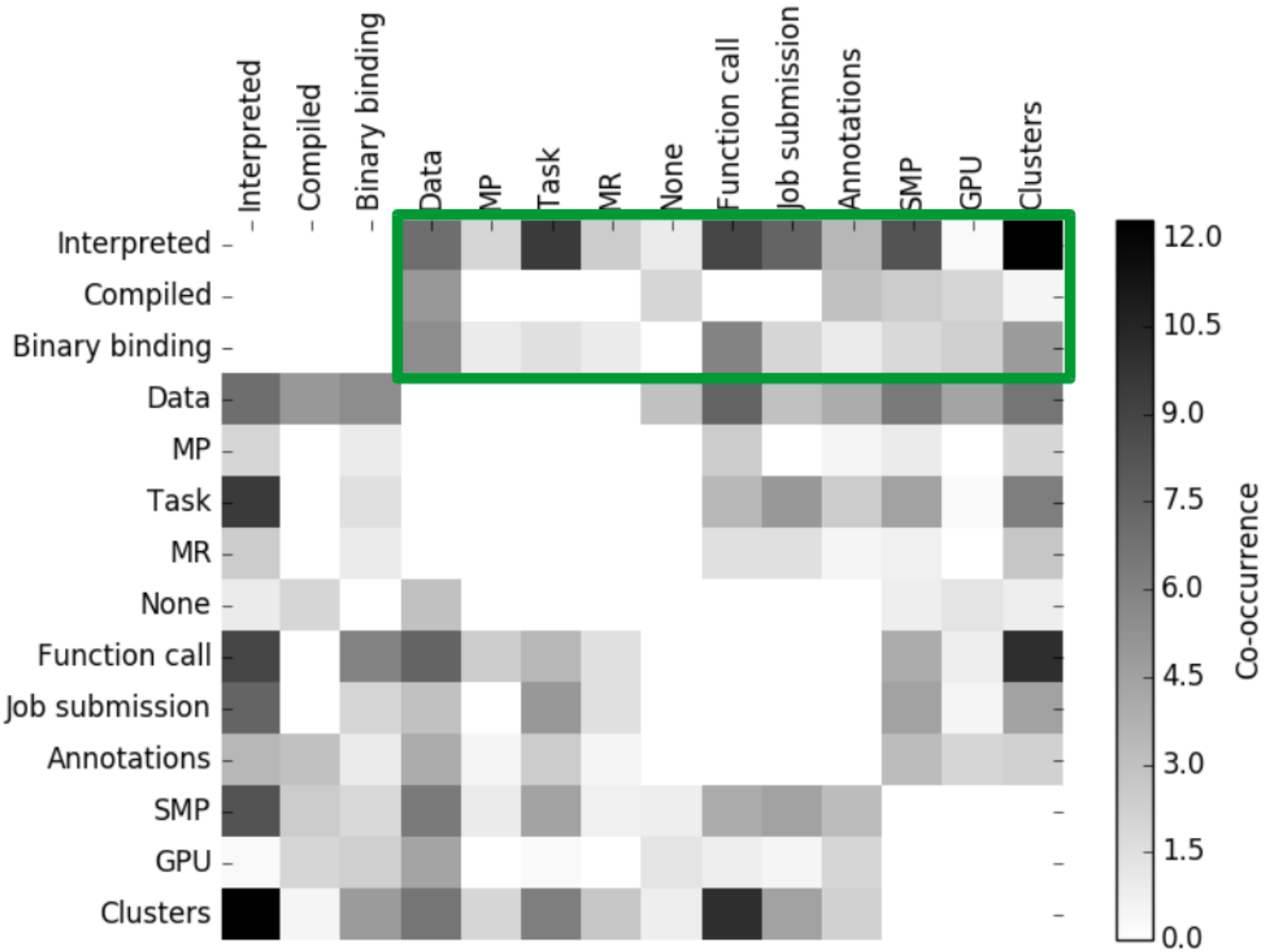
Co-occurrence matrix (0)



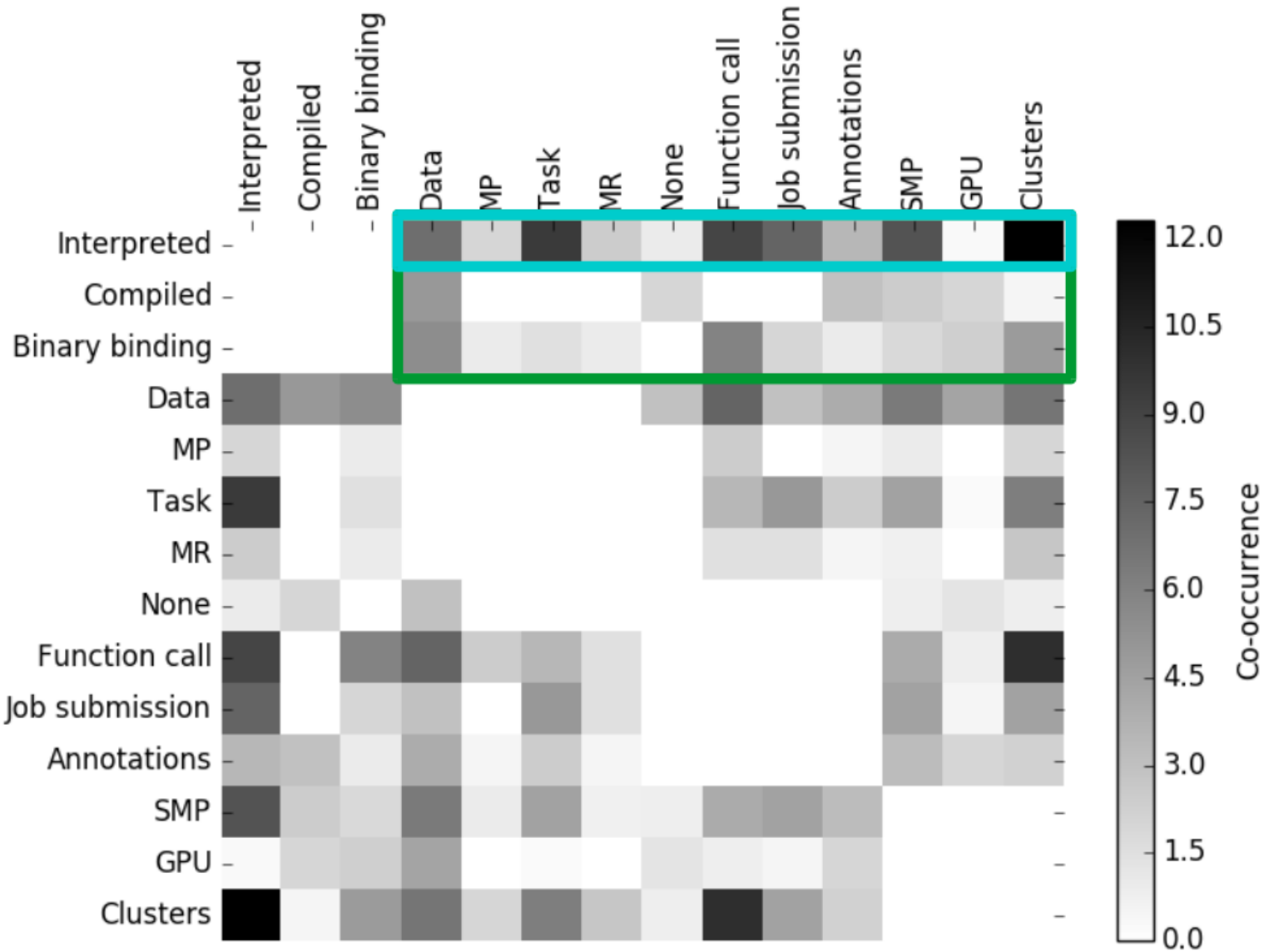
Co-occurrence matrix (1)



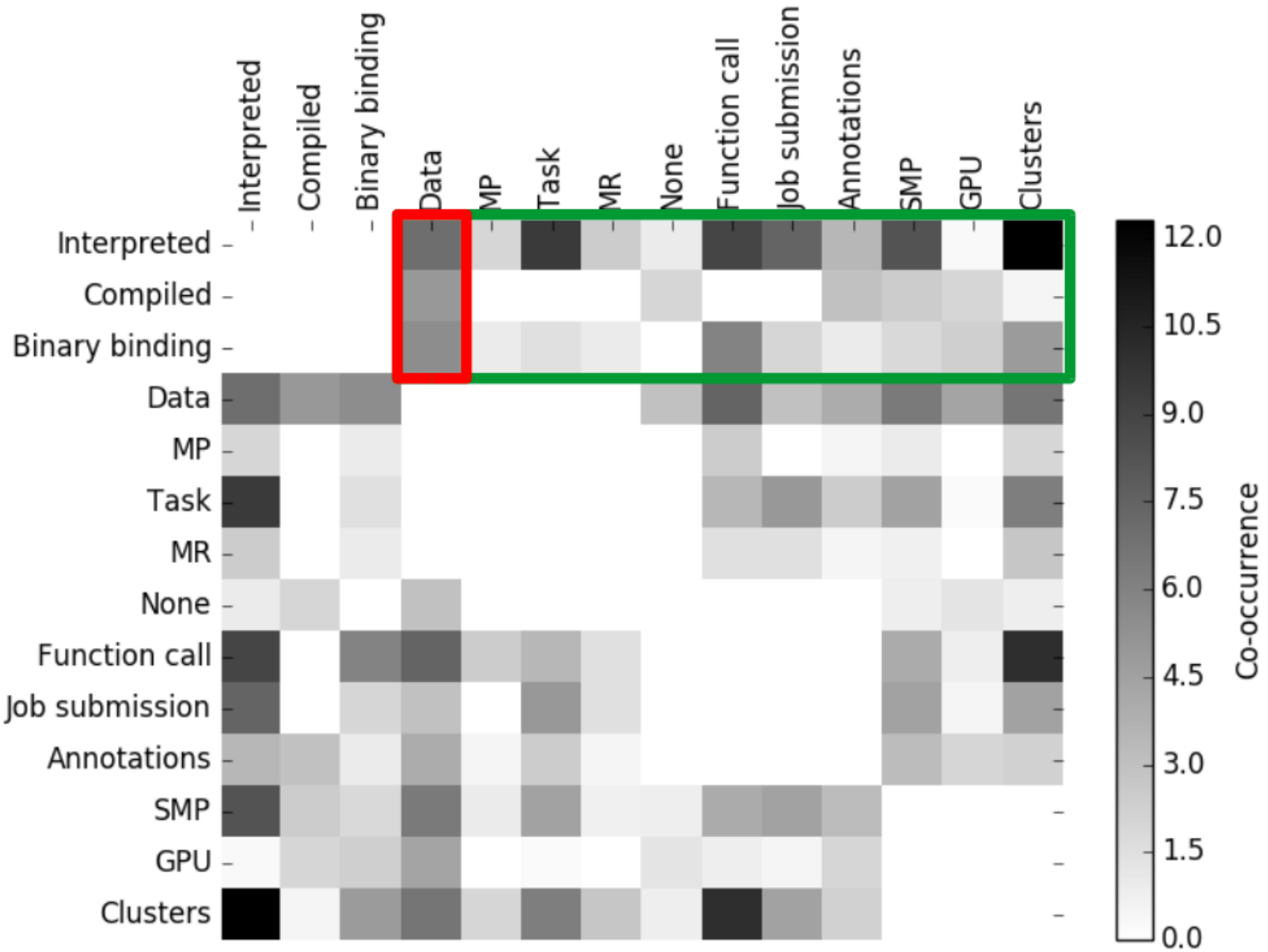
Co-occurrence matrix (2)



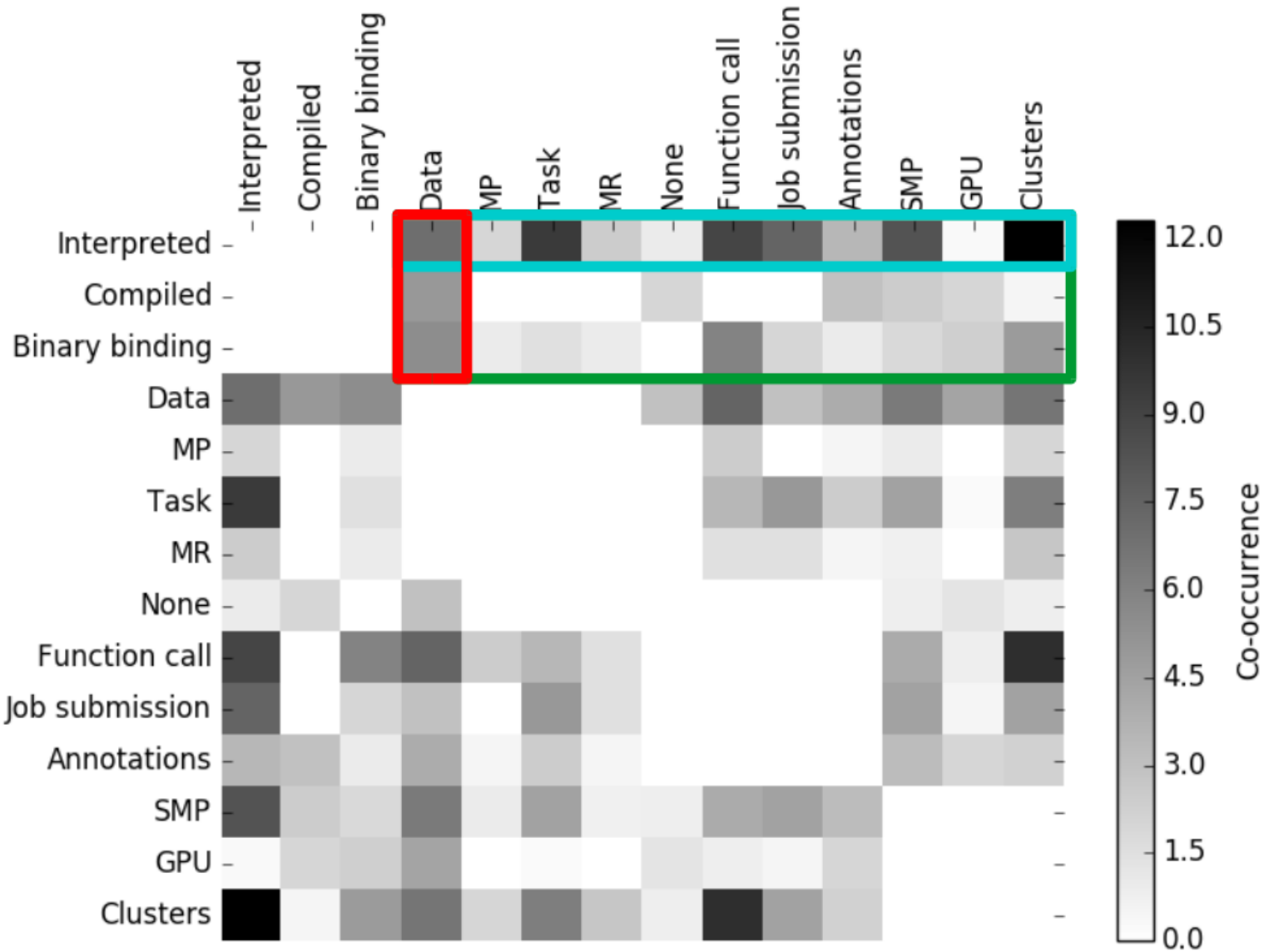
Co-occurrence matrix (3)



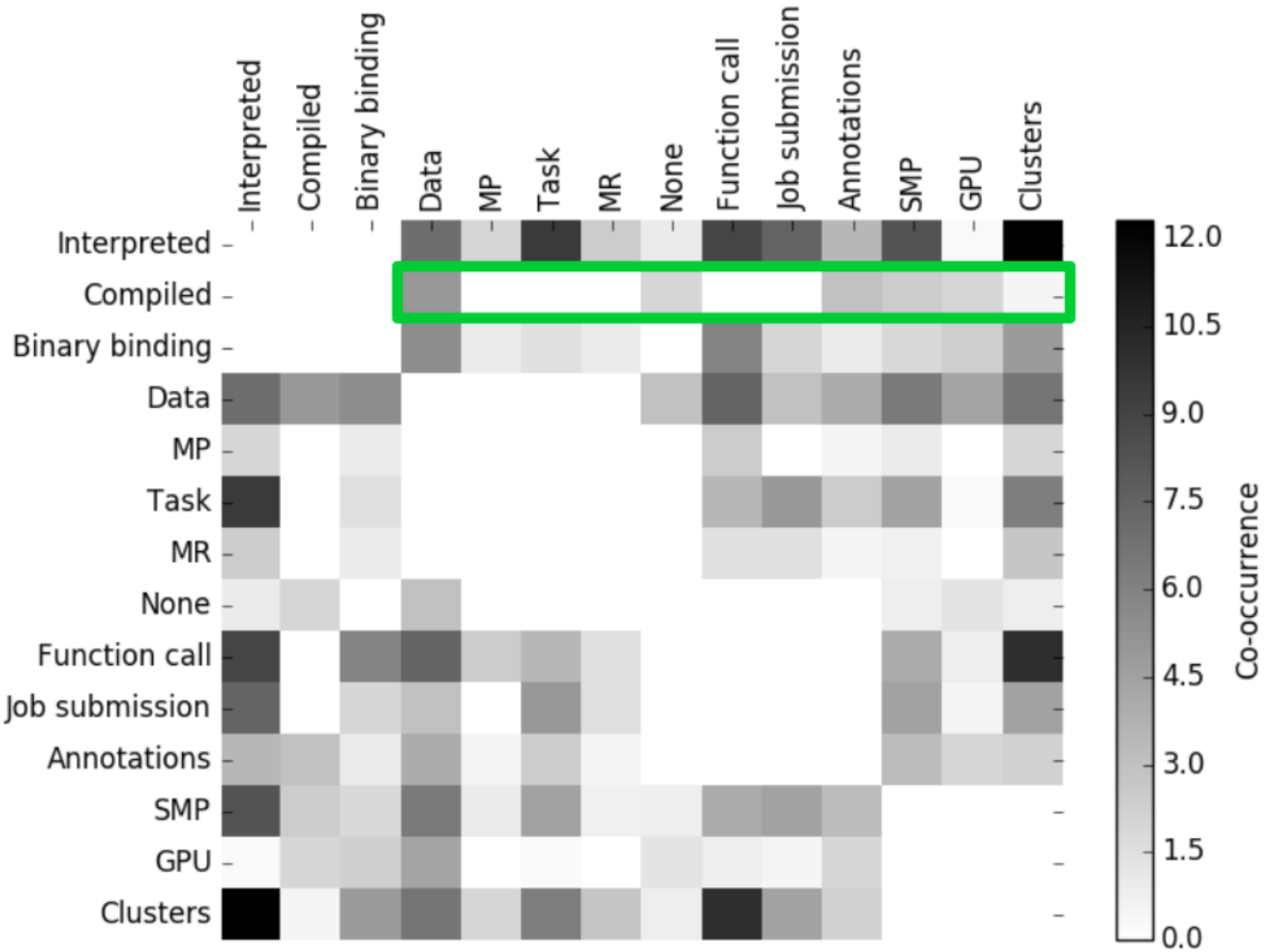
Co-occurrence matrix (4)



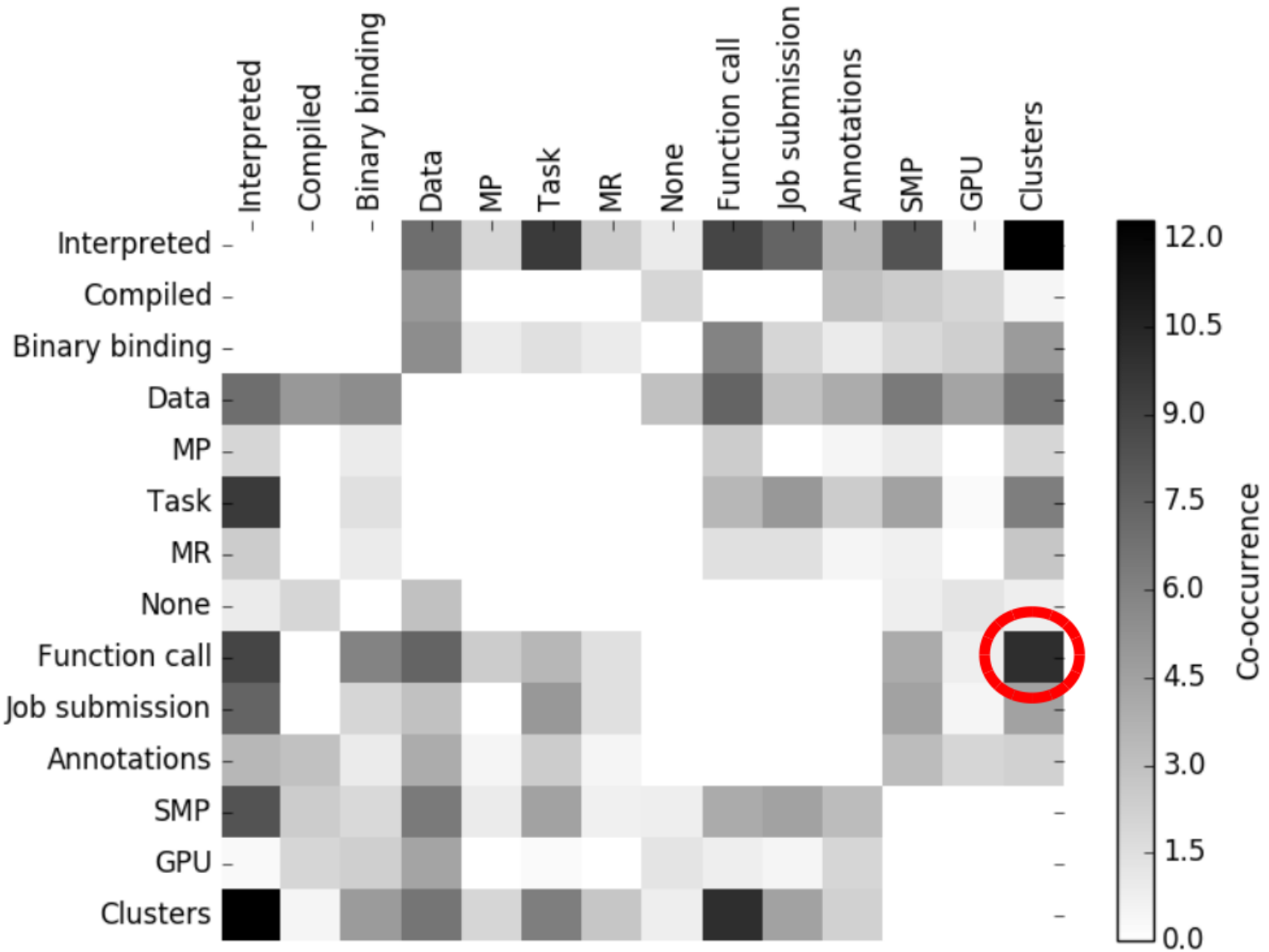
Co-occurrence matrix (5)



Co-occurrence matrix (6)



Co-occurrence matrix (7)



Conclusions

1. Parallel Python is a promising research field.
2. Community has given a plethora of alternatives, this work presents a handy comparative analysis.

Conclusions (1)

3. The most common characteristics are:
   - Using Python interpreter to run the code or using external code.
   - To offer data parallelism or task parallelism.
   - Majority of projects support full language, except compiled projects.
   - Parallelism is exposed as function calls or explicit job submission.
   - Targeting SMP and clusters platforms is common.

Guillermo Cornejo Suárez - GmoCornejoS@gmail.com