Automatic task-based parallelization of Python codes

Cristián Ramón-Cortés
Ramon Amela
Jorge Ejarque
Philippe Clauss
Rosa M. Badia

MS12: Task-based Programming for Scientific Computing: Runtime Support

Outline

- Introduction
  - PLUTO
  - PyCOMPSs
- AutoParallel
  - Annotation
  - Architecture
- Evaluation
- Conclusions and Future Work

Introduction

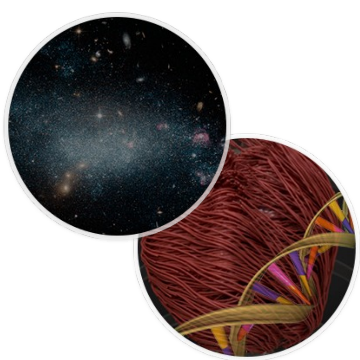
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Motivation

Parallel Issues
- Identifying parallel regions
- Concurrency management
- Execution orchestration

Distributed Issues
- Remote execution
- Data transfers

Ease the development of distributed applications



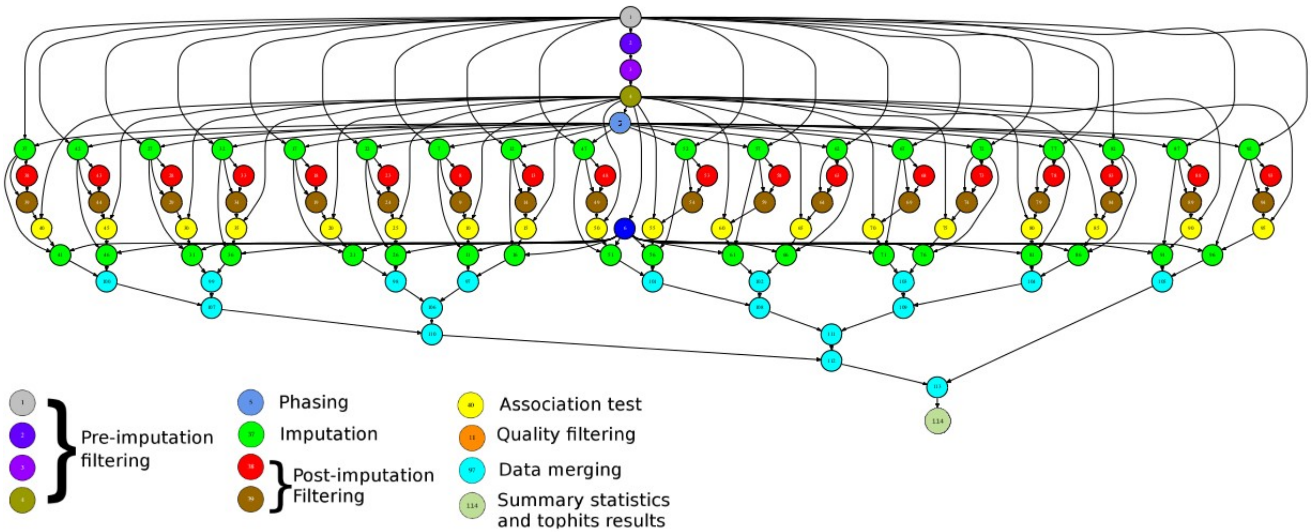
THE GOAL:

Any field expert can scale up an application to hundreds of cores

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

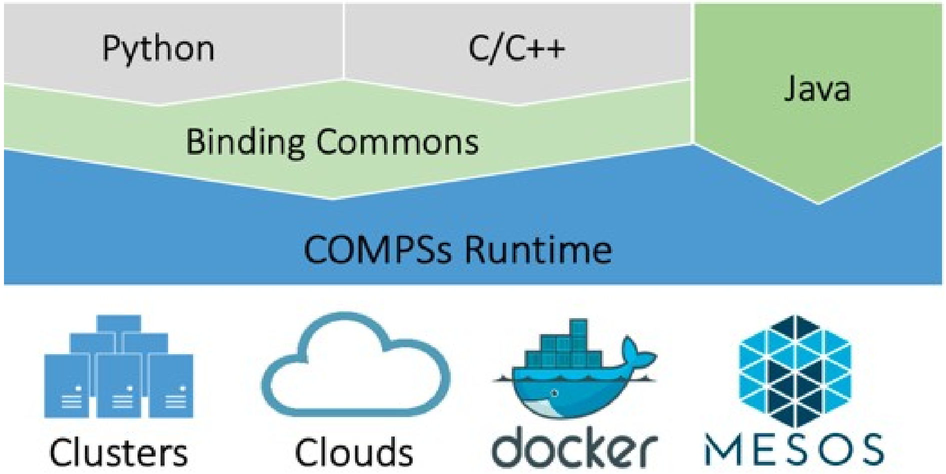
COMPSs

- Based on sequential programming
  - General purpose programming language + annotations
- Task-based programming model
  - Task is the unit of work
  - Implicit Workflow: Builds a task graph at runtime that expresses potential concurrency



COMPSs

- Infrastructure agnostic
  - Same application runs on clusters, grids, clouds and containers



- Supports other types of parallelism
  - Multi-threaded tasks (i.e., MKL kernels)
  - Multi-node tasks (i.e., MPI applications)
  - Non-native tasks (i.e., binaries)
  - Nested PyCOMPSs applications
  - Integration with BSC OmpSs

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PyCOMPSs Annotation

- Python decorators for task selection + synchronization API
  - Instance and class methods
  - Task data directions

```
@task(a=IN, b=IN, c=INOUT)
def multiply_acum(a, b, c):
    c += a * b
```

```
@task(returns=int)
def multiply(a, b, c):
    return c + a * b
```

```
@constraint(computingUnits="2")
@task(file=FILE_IN)
def my_task(x):
    ...
```

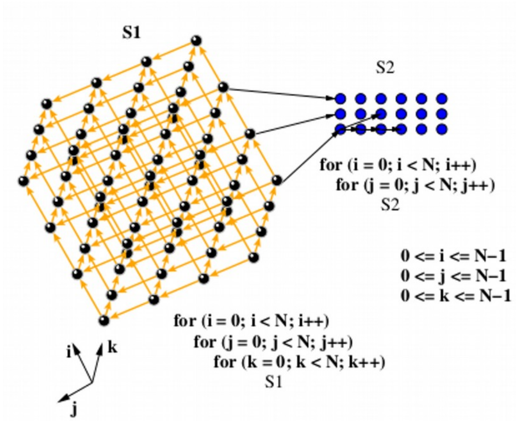
```
@binary(binary="sed")
@task(f=FILE_INOUT)
def binary_task(flag, expr, f):
    pass
```

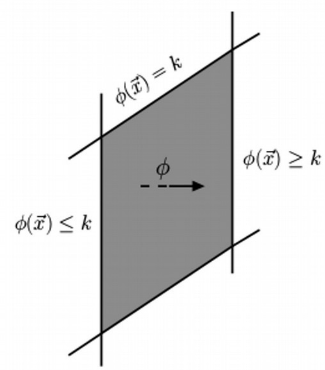
```
@task(returns=dict)
def wordcount(block):
  ...

@task(result=INOUT)
def reduce(result, pres):
  ...

def main(a, b, c):
  for block in data:
    pres = wordcount(block)
    reduce(result, pres)
  result = compss_wait_on(result)

  # f = compss_open(fn)
  # compss_delete_file(f)
  # compss_delete_object(o)
  # compss_barrier()
```

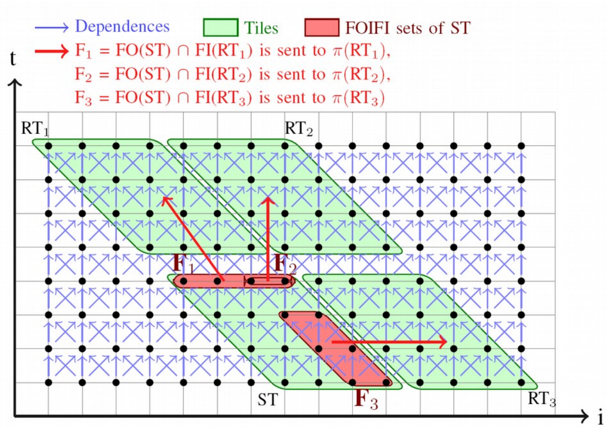
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PLUTO

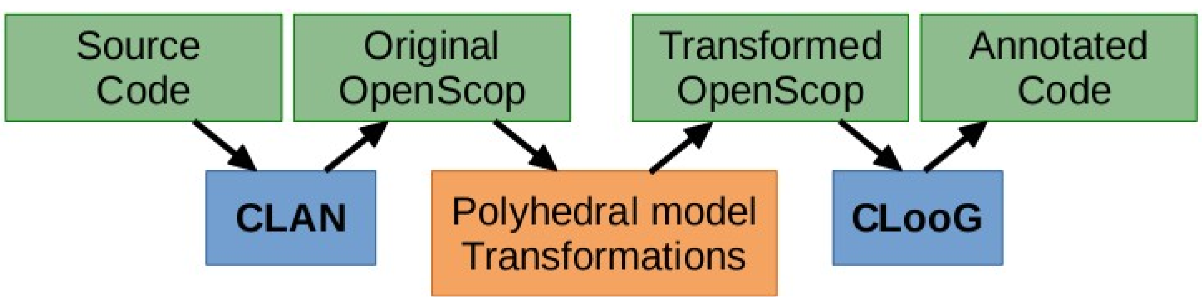
- The **Polyhedral Model** represents the instances of the loop nests' statements as integer points inside a polyhedron







- **PLUTO** is an automatic parallelization tool based on the Polyhedral Model to optimize arbitrarily nested loop sequences with affine dependencies



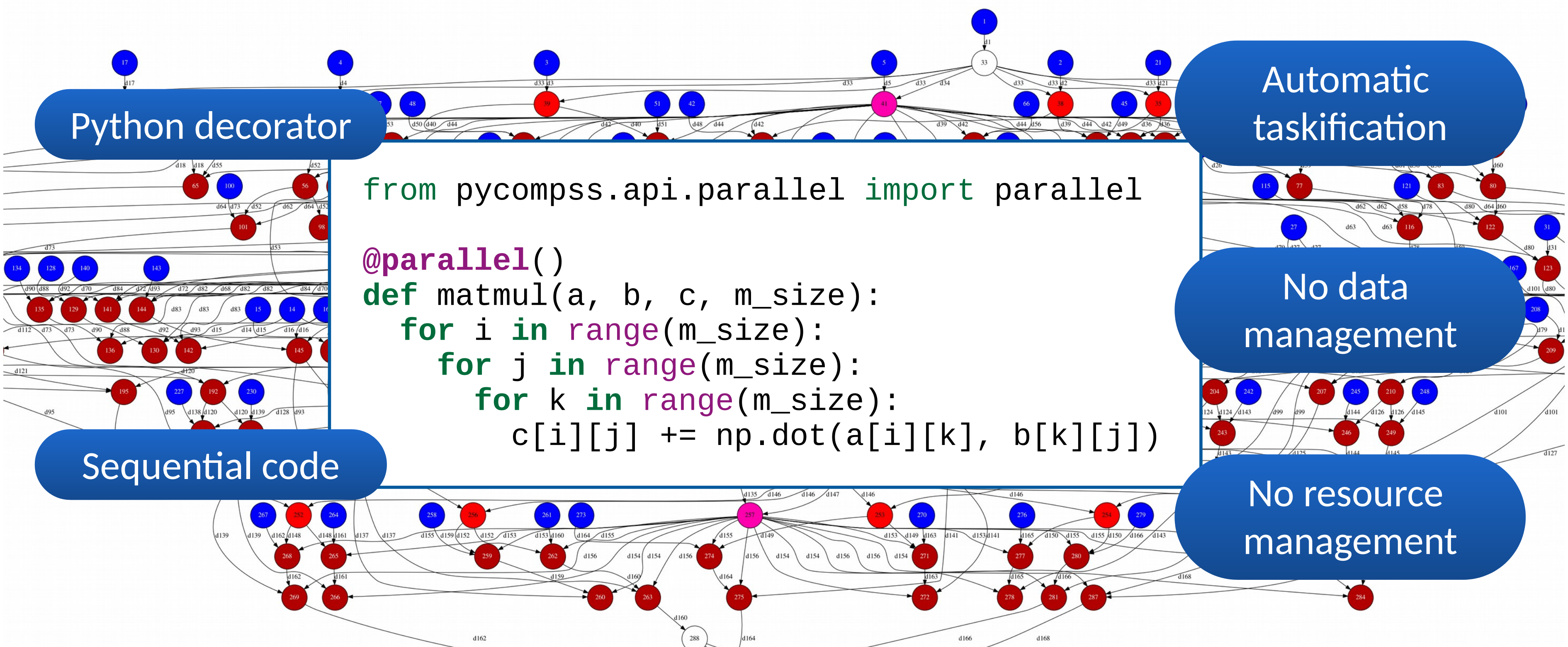
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

AutoParallel

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

AutoParallel

A single Python decorator to parallelize and distributedly execute sequential code containing affine loop nests



Python decorator

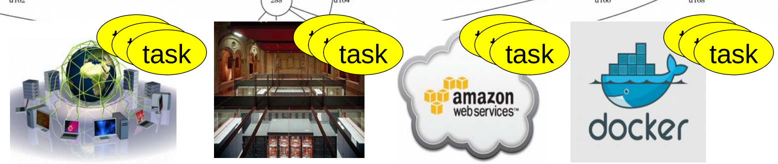
Automatic taskification

```python
from pycompss.api.parallel import parallel

@parallel()
def matmul(a, b, c, m_size):
  for i in range(m_size):
    for j in range(m_size):
      for k in range(m_size):
        c[i][j] += np.dot(a[i][k], b[k][j])
```

No data management

Sequential code

No resource management



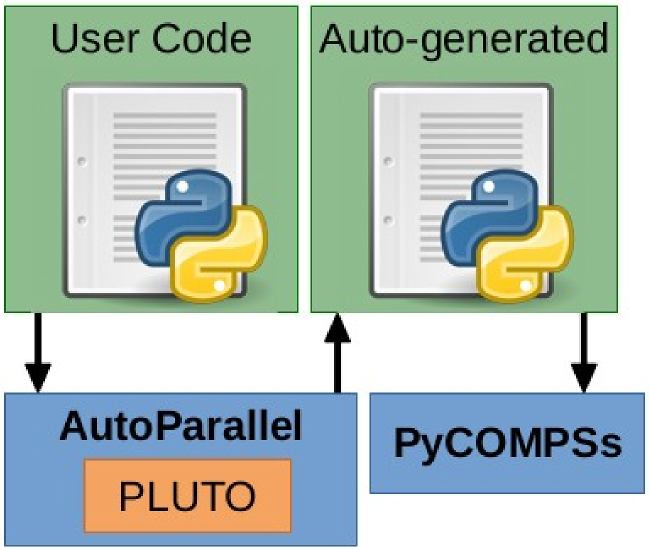
Grid Cluster Cloud Container

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

AutoParallel Annotation

- Taskification of affine loop nests at runtime

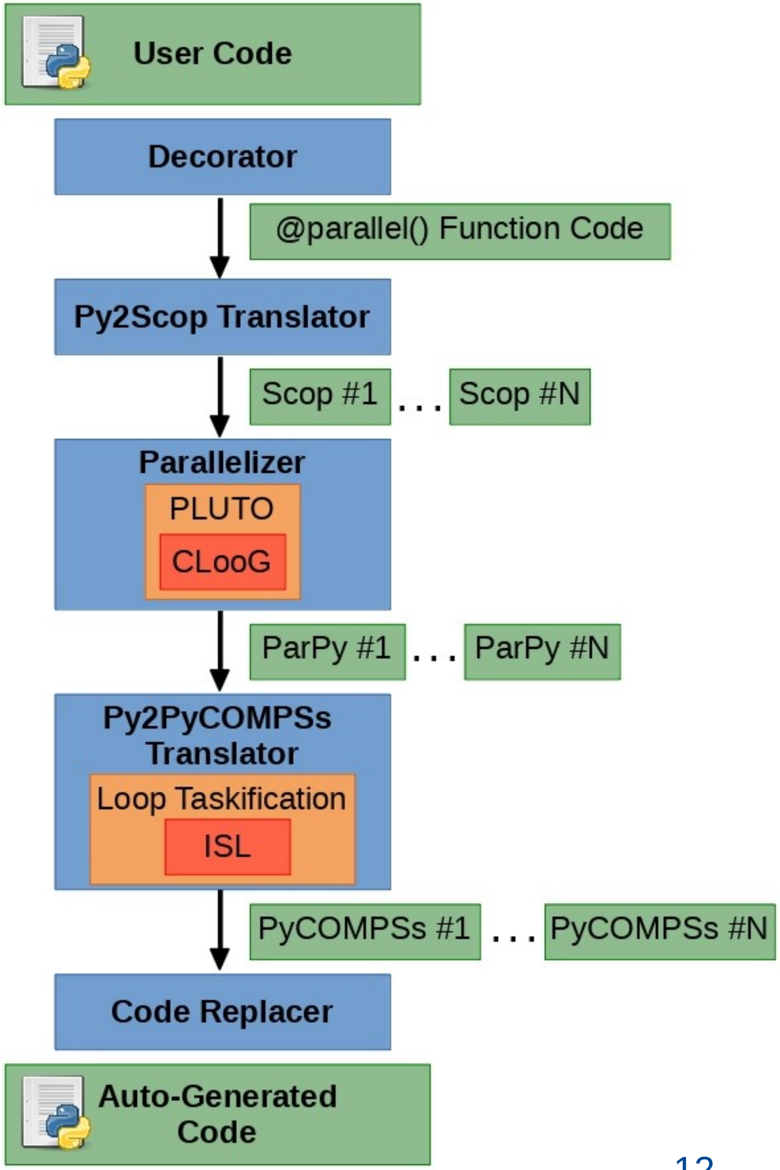
```
@parallel()
def matmul(a, b, c, m_size):
  for i in range(m_size):
    for j in range(m_size):
      for k in range(m_size):
        c[i][j] += np.dot(a[i][k], b[k][j])
```




```
# [COMPSs AutoParallel] Begin Autogenerated code
@task(var2=IN, var3=IN, var1=INOUT)
def S1(var2, var3, var1):
  var1 += np.dot(var2, var3)

def matmul(a, b, c, m):
  if m >= 1:
    for t1 in range(0, m - 1): #i
      lbp = 0
      ubp = m - 1
      for t2 in range(lbp, ubp + 1): #k
        lbv = 0
        ubv = m - 1
        for t3 in range(lbv, ubv + 1): #j
          S1(a[t1][t2], b[t2][t3], c[t1][t3])
  compss_barrier()
# [COMPSs AutoParallel] End Autogenerated code
```

AutoParallel Architecture

- Decorator
  - Implements the @parallel decorator
- Python to OpenScop translator
  - Builds a Python Scop object from the Python's AST representing each affine loop nest detected in the user function
- Parallelizer
  - Parallelizes an OpenScop file and returns its Python code using OpenMP syntax
- Python to PyCOMPSs translator
  - Inserts the PyCOMPSs syntax (task annotations and data synchronizations) to the annotated Python code (uses Python's AST)
- Code replacer
  - Replaces each loop nest in the initial user code by the auto-generated code



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Evaluation

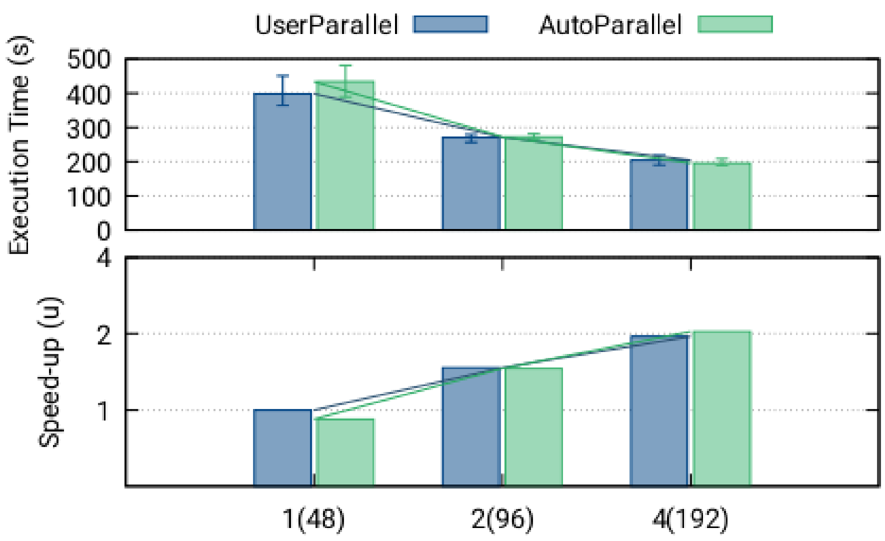
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Cholesky

	Code Analysis		
	LoC	*CC*	*NPath*
User	220	26	112
Auto	274	36	14.576

LoC	Lines Of Code
CC	Cyclomatic Complexity
NPath	Npath Complexity

	Loop Analysis		
	#Main	*#Total*	*Depth*
User	1	4	3
Auto	3	9	3



	Problem Size			Execution		
	Total Matrix Size	*#Blocks*	*Block Size*	*Task Types*	*#Tasks*	*SpeedUp @ 192 cores*
User	65.536 x 65.536	32 x 32	2048 x 2048	3	6.512	1,95
Auto				4	7.008	2,04

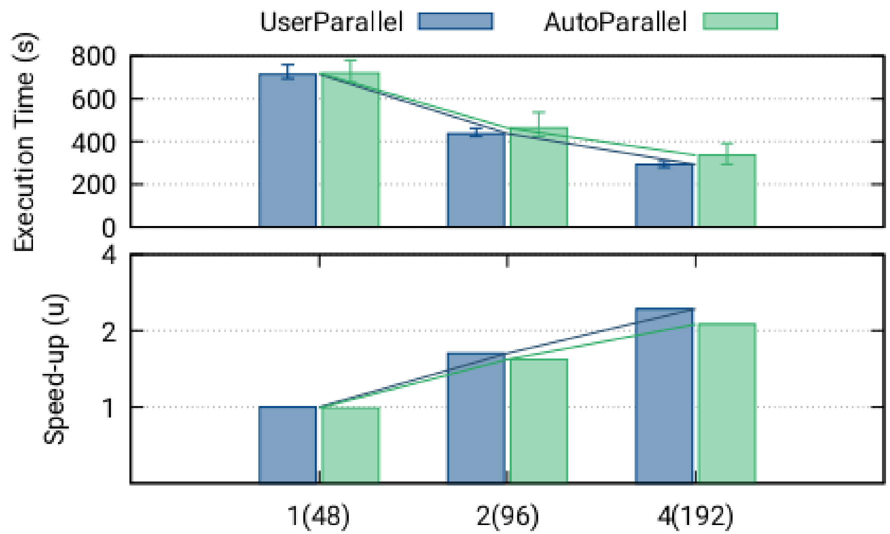
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

LU

	Code Analysis		
	LoC	***CC***	***NPath***
User	238	35	79.872
Auto	320	39	331.776

	Loop Analysis		
	#Main	***#Total***	***Depth***
User	2	6	3
Auto	2	6	3

LoC Lines Of Code
CC Cyclomatic Complexity
NPath Npath Complexity



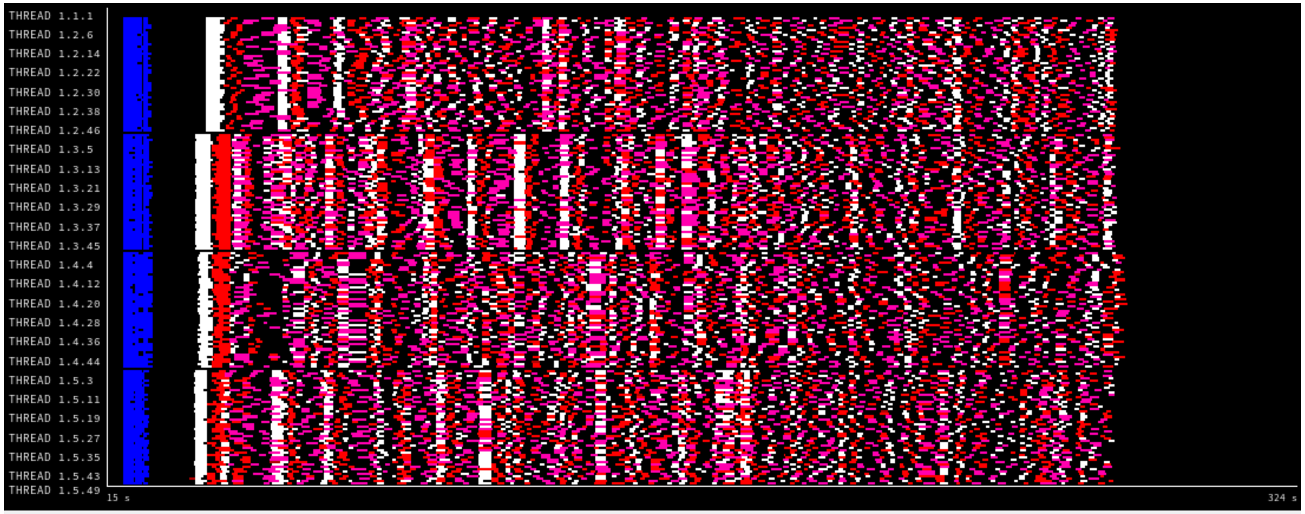
	Problem Size			Execution		
	Total Matrix Size	***#Blocks***	***Block Size***	***Task Types***	***#Tasks***	***SpeedUp @ 192 cores***
User	49.152 x 49.152	24 x 24	2048 x 2048	4	14.676	2,45
Auto				12	15.227	2,13

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

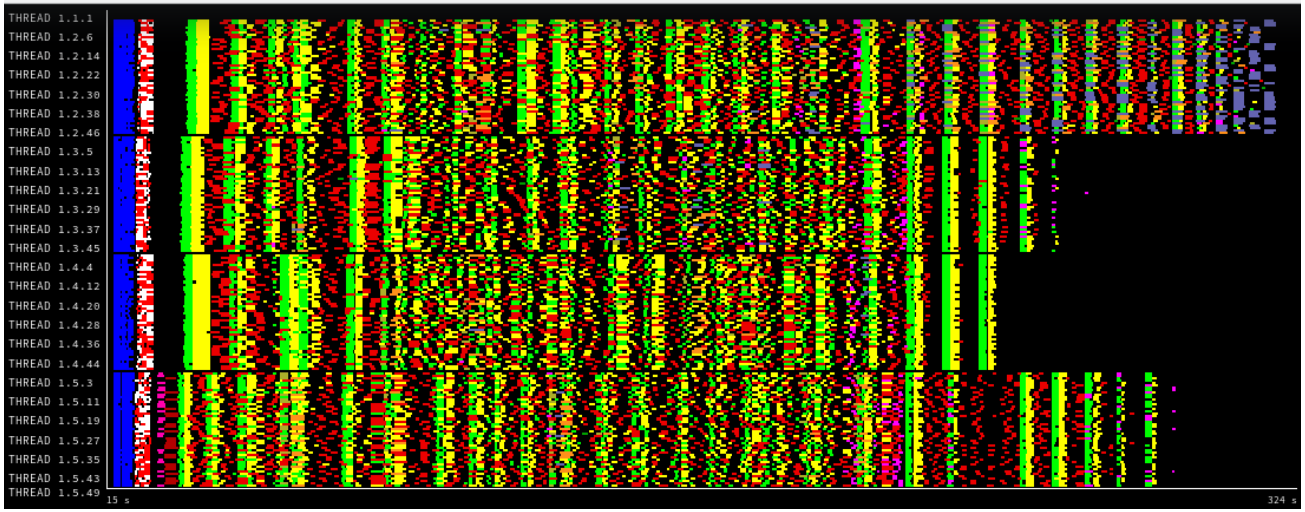
LU

- In-depth performance analysis
  - Paraver trace with 4 workers (192 cores)

UserParallel



AutoParallel

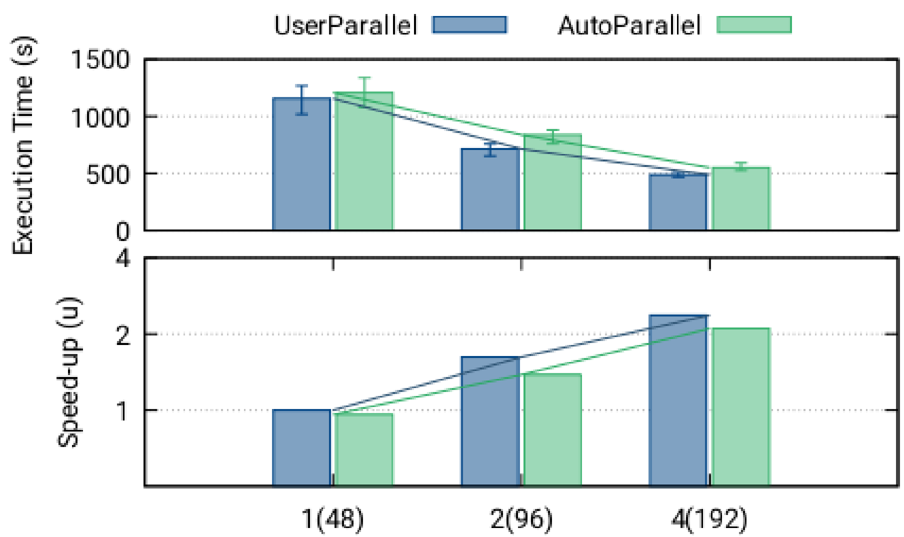


QR

LoC	Lines Of Code
CC	Cyclomatic Complexity
NPath	Npath Complexity

	Code Analysis		
	LoC	*CC*	*NPath*
User	303	41	168
Auto	406	43	344

	Loop Analysis		
	#Main	*#Total*	*Depth*
User	1	6	3
Auto	2	7	3



	Problem Size			Execution		
	Total Matrix Size	*#Blocks*	*Block Size*	*Task Types*	*#Tasks*	*SpeedUp @ 192 cores*
User	32.768 x 32.768	16 x 16	2048 x 2048	4	19.984	2,37
Auto				20	26.304	2,10

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Conclusions and Future Work

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Conclusions and Future Work

- **AutoParallel goes one step further in easing the development of distributed applications**
  - It is a Python module to automatically parallelize affine loop nests and execute them in distributed infrastructures
  - The evaluation shows that the automatically generated codes for the Cholesky, LU, and QR applications can achieve the same performance than the manually parallelized versions

- **Next steps**
  - Loop taskification: An automatic way to create blocks from sequential applications based on loop tiles. Requires:
    - Research on how to simplify the chunk accesses from the AutoParallel module
    - Extend PyCOMPSs to support collection objects (e.g., lists)
  - Integration with different tools similar to PLUTO to support a larger scop of loop nests (e.g., APOLLO)

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

ExaQUte

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Thank you

cristianrcv/pycompss-autoparallel

http://compss.bsc.es/

cristian.ramon-cortes@bsc.es