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
**Motivation**

**THE GOAL:**
Any field expert can scale up an application to hundreds of cores

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

- **Distributed Issues**
  - Remote execution
  - Data transfers

**Ease the development of distributed applications**
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
PyCOMPSs Annotation

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

```python
@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
```

```python
@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()
```
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.
AutoParallel
AutoParallel

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

```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])
```

Grid        Cluster     Cloud    Container

Automatic taskification

Python decorator

Sequential code

No data management

No resource management

10
AutoParallel Annotation

- Taskification of affine loop nests at runtime

```python
@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])
```

```python
# [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
Evaluation
Cholesky

<table>
<thead>
<tr>
<th>Code Analysis</th>
<th>LoC</th>
<th>CC</th>
<th>NPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>220</td>
<td>26</td>
<td>112</td>
</tr>
<tr>
<td>Auto</td>
<td>274</td>
<td>36</td>
<td>14.576</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loop Analysis</th>
<th>#Main</th>
<th>#Total</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Auto</td>
<td>3</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Matrix Size</strong></td>
<td><strong>#Blocks</strong></td>
</tr>
<tr>
<td>User</td>
<td>65.536 x 65.536</td>
</tr>
<tr>
<td>Auto</td>
<td>65.536 x 65.536</td>
</tr>
</tbody>
</table>

LoC: Lines Of Code  
CC: Cyclomatic Complexity  
NPath: Npath Complexity
<table>
<thead>
<tr>
<th></th>
<th>LoC</th>
<th>CC</th>
<th>NPath</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code Analysis</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>238</td>
<td>35</td>
<td>79.872</td>
</tr>
<tr>
<td>Auto</td>
<td>320</td>
<td>39</td>
<td>331.776</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>#Main</th>
<th>#Total</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loop Analysis</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Auto</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th><strong>Problem Size</strong></th>
<th><strong>Execution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Total Matrix Size</strong></td>
<td><strong>#Blocks</strong></td>
</tr>
<tr>
<td>User</td>
<td>49.152 x 49.152</td>
<td>24 x 24</td>
</tr>
<tr>
<td>Auto</td>
<td>12</td>
<td>15.227</td>
</tr>
</tbody>
</table>
LU

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

<table>
<thead>
<tr>
<th></th>
<th>LoC</th>
<th>CC</th>
<th>NPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>303</td>
<td>41</td>
<td>168</td>
</tr>
<tr>
<td>Auto</td>
<td>406</td>
<td>43</td>
<td>344</td>
</tr>
</tbody>
</table>

### Loop Analysis

<table>
<thead>
<tr>
<th></th>
<th>#Main</th>
<th>#Total</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Auto</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

### Problem Size

<table>
<thead>
<tr>
<th></th>
<th>Total Matrix Size</th>
<th>#Blocks</th>
<th>Block Size</th>
<th>Task Types</th>
<th>#Tasks</th>
<th>SpeedUp @ 192 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>32.768 x 32.768</td>
<td>16 x 16</td>
<td>2048 x 2048</td>
<td>4</td>
<td>19.984</td>
<td>2.37</td>
</tr>
<tr>
<td>Auto</td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td>26.304</td>
<td>2.10</td>
</tr>
</tbody>
</table>

---

**LoC** Lines Of Code  
**CC** Cyclomatic Complexity  
**NPath** Npath Complexity
Conclusions and Future Work
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)
Thank you

cristianrcv/pycompss-autoparallel

http://compss.bsc.es/

cristian.ramon-cortes@bsc.es