ExaQUte

Exascale Quantification of Uncertainties for Technology and Science Simulation

D5.2 Release of ExaQUte MLMC Python engine

Document information table

Contract number:	800898
Project acronym:	ExaQUte
Project Coordinator:	CIMNE
Document Responsible Partner:	EPFL
Deliverable Type:	Other (software)
Dissemination Level:	PUblic
Related WP & Task:	WP5 task 5.2
Status:	Approved

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **800898**

Authoring

Prepared by EPFL, CIMNE and BSC				
Authors	Partner	Modified	Version	Comments
Ramon Amela	BSC	All	1.0.0	PyCOMPSs expertise
Quentin Ayoul-Guilmard	EPFL			Development
Sundar Ganesh				
Riccardo Tosi	CIMNE			
Rosa M. Badia	BSC			Supervision
Fabio Nobile	EPFL			
Riccardo Rossi	CIMNE			

Change Log

Versions	Modified Page/Sections	Comments
1.0.0	All	Submitted version

Approval

Approved by CIMNE and EPFL				
	Name	Partner	Date	OK
Task leader	Fabio Nobile	EPFL	2019-05-30	✓
Coordinator	Riccardo Rossi	CIMNE	2019-05-30	✓

Executive summary

In this deliverable, the ExaQUte XMC library is introduced. This report is meant to serve as a supplement to the publicly release of the library. In the following sections, the ExaQUte XMC library is described along with its current and future capabilities. The structure of the library, along with its dynamic import mechanism, are described using samples of code. The algorithms behind the example files supplied with the public release are explained in detail as well.

Contents

1 Introduction

The ExaQUte XMC library [released as 1] offers the user the ability to carry out uncertainty quantification simulations using a multitude of Monte Carlo algorithms. Users of the library will eventually be able to select one of many pre-programmed algorithms such as

1. simple Monte Carlo;
2. adaptive Monte Carlo;
3. simple multi-level Monte Carlo;
4. adaptive multi-level Monte Carlo;
5. continuation multi-level Monte Carlo;
6. multi-index Monte Carlo;
7. multi-fidelity Monte Carlo.

They will also be able to chose from pre-programmed interfaces with widely used numerical analysis packages. The library also offers parallelisation capabilities using the PyCOMPSs [see 2, 4, 6] distributed computing framework for use in high-performance machines.

The library is also programmed in a modular way that allows users to construct their own 'X' Monte Carlo (XMC) algorithm from several building block functions. It also allows them to write custom-interfaces easily for their own numerical analysis packages such as custom-finite element analysis or computational fluid dynamics libraries.

2 ExaQUte XMC library structure

The ExaQUte XMC library is organised as follows. The global directory `xmc` is a python package that contains all the packages, subpackages and modules of the library. Within this folder, each file contains the definition of one class. Every file is treated as a module that can be imported. If the class is called `ClassName`, the file is named `className.py`

Each class contains multiple types of members. They are as follows

1. Instances of classes
2. Instances of function definition objects

3. Variables and other data structures

4. Method definitions

The instantiation of function objects is used to enable a function to have multiple definitions. For example, we require that the `optimalIndexSet` method of the `HierarchyOptimiser` class to be called as `optimalIndexSet()`, but to evaluate different expressions for the optimal number of levels based on the type of XMC algorithm used.

In this way the user can simply specify the type of algorithm they would like to run and the definition of `optimalIndexSet` automatically changes based on this, without any further intervention. This is achieved by instantiating `optimalIndexSet` dynamically to a specific function definition through the constructor of the `HierarchyOptimiser` class as follows.

```
# hierarchyOptimiser.py
class HierarchyOptimiser():
    def __init__(self, **keywordArgs):
        ...
        self.optimalIndexSet =
            dynamicImport(keywordArgs.get("optimalIndexSet"))
        ...
```

The `dynamicImport` method sets the definition of `optimalIndexSet` based on inputs to the constructor. In this way, every call to `optimalIndexSet` in the code is replaced by a call to the specific definition.

It is hence implied that every "general" method such as `optimalIndexSet` will have a corresponding list of specific definitions, one of which is selected at runtime during the construction of the class containing the method.

To organise this, every `className.py` file has a corresponding folder by the name `methodDefs_className`. Inside this folder, there are multiple files named `generalMethod.py`, one for each member of `ClassName` that is a function object instance of the class. Within each `generalMethod.py`, there are a list of definitions as follows -

```
# generalMethod.py
def specificDefinition1():
    ...

def specificDefinition2():
    ...
```

For the example in this section, the instantiation then occurs as follows.

```
# elsewhere.py
if (xmcAlgorithmType == "xmcAlgorithm1"):
```

```
    keywordArgs["optimalIndexSet"] = "xmc.
       methodDefs_hierarchyOptimiser.optimalIndexSet.
       specificDefinition1"

x = HierarchyOptimiser(**keywordArgs)
```

3 Instructions to run tutorial files

All tutorial files are to be run as follows

```
$ python3 example_*.py
```

No additional arguments are required. If the user has the PyCOMPSs scheduling tool installed and the imports in the required files appropriately set, then the example files can be run with the following command.

```
$ runcompss --options=value /global/path/to/example_*.py
```

However, the user must note that the appropriate imports have to be changed in places marked through the program. This functionality will be automated in a future release.

4 Tutorial file and algorithm description

In the following subsections, we describe the algorithms currently available in the example cases provided with the library. `example_*_Kratos.py` contains the same algorithm as the corresponding `example_*.py`, but a partial differential equation is solved for the quantity of interest in the Kratos solver [released as 5] instead of a simple random number generator.

4.1 Simple Monte Carlo

`example_mc.py` contains a simple Monte Carlo algorithm that estimates the mean of a standard normal variable. The algorithm begins with a prescribed number of samples. It estimates the mean and the error in the mean estimate. If the error does not satisfy a prescribed tolerance, it doubles the number of samples and repeats the process until convergence.

4.2 Adaptive Monte Carlo

`example_amc.py` contains an adaptive Monte Carlo algorithm that estimates the mean of a standard normal variable. The algorithm begins with a prescribed number of samples. It estimates the mean and the error in the mean

estimate. If the error does not satisfy a final tolerance, it then computes a new number of samples that will be required to satisfy the tolerance requirement based on the assumption of asymptotic normality, and repeats this procedure until convergence.

4.3 Simple Multi-Level Monte Carlo

`example_mlmc.py` contains a simple Multi-Level Monte Carlo algorithm that estimates the mean of a standard normal variable. A hierarchy of normal random generators whose, the mean and variance of the differences of whose samples decay geometrically with increasing level towards 0, are used to generate samples. The algorithm begins with a prescribed number of samples and levels. It estimates the mean and the error in the mean estimate. If the error does not satisfy a prescribed tolerance, it increases the number of levels by one, doubles the number of samples in all existing levels, and adds a default number of samples to the new level. This process is repeated until convergence.

4.4 Adaptive Multi-Level Monte Carlo

`example_amlmc.py` contains an adaptive Multi-Level Monte Carlo algorithm[detailed in 3] that estimates the mean of a standard normal variable. A hierarchy of normal random generators whose, the mean and variance of the differences of whose samples decay geometrically with increasing level towards 0, are used to generate samples. The algorithm begins with a prescribed number of samples and levels. It estimates the mean and the error in the mean estimate. It then fits geometric models for the bias, variance and cost across levels on the computed data. If the error does not satisfy a prescribed tolerance, it increases the number of levels by one and computes the cost-optimal number of samples for each level using level-wise bias, variance and cost estimates as well as the models for these three quantities. to the new level. This process is repeated until convergence.

References

[1] Ramon Amela, Quentin Ayoul-Guilmard, Rosa M Badia, Sundar Ganesh, Fabio Nobile, Riccardo Rossi and Riccardo Tosi. *ExaQUte XMC*. Comp. software. Version 1.0.0. ExaQUte consortium, May 2019. DOI: 10.5281/zenodo.3235833.

[2] Rosa M. Badia, J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes and R. Sirvent. 'COMP Superscalar, an interoperable programming framework'. In: *SoftwareX* 3–4 (Dec. 2015). DOI: 10.1016/j.softx.2015.10.004.

[3] Michael B. Giles. 'Multilevel Monte Carlo methods'. In: *Acta Numerica* 24 (2015), pp. 259–328. DOI: 10.1017/S096249291500001X.

[4] F. Lordan et al. 'ServiceSs: an interoperable programming framework for the Cloud'. In: *Journal of Grid Computing* 12 (1 Mar. 2014). DOI: 10.1007/s10723-013-9272-5.

[5] Riccardo Rossi, Carlos Roig, Marc Núñez, Riccardo Tosi, Brendan Keith, Rosa M Badia and Ramon Amela. *KratosMultiphysics/Kratos: Exaqute M12*. Comp. software. Version 7.0-Exaqute. ExaQUte consortium, May 2019. DOI: 10.5281/zenodo.3234645.

[6] Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M. Badia, Jordi Torres, Toni Cortes and Jesús Labarta. 'PyCOMPSs: Parallel computational workflows in Python'. In: *International Journal of High Performance Computing Applications* 31 (1 2017). DOI: 10.1177/1094342015594678.