H2020 FETHPC-1-2014

An Exascale Programming, Multi-objective Optimisation and Resilience Management Environment Based on Nested Recursive Parallelism

Project Number 671603

D3.1 – API Implementation for Recursive Parallelism (a)

WP3: High-level parallel API and API-aware optimizing source-to-source compiler

Version: 1.0

Author(s): Herbert Jordan (UIBK)

Date: 24/03/16

Due date:	PM6
Submission date:	day/month/year
Project start date:	01/10/2015
Project duration:	36 months
Deliverable lead organization	UIBK
Version:	1.0
Status	Final
Author(s):	Herbert Jordan (UIBK)
Reviewer(s)	Kostas Katrinis (IBM), Roman Iakymchuk (KTH)

Dissemination level	
PU	*Public*

Disclaimer

This deliverable has been prepared by the responsible Work Package of the Project in accordance with the Consortium Agreement and the Grant Agreement Nr 671603. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the Project and to the extent foreseen in such agreements.

Acknowledgements

The work presented in this document has been conducted in the context of the EU Horizon 2020. AllScale is a 36-month project that started on October 1st, 2015 and is funded by the European Commission.

The partners in the project are UNIVERSITÄT INNSBRUCK (UBIK), FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN NÜRNBERG (FAU), THE QUEEN'S UNIVERSITY OF BELFAST (QUB), KUNGLIGA TEKNISKA HÖGSKOLAN (KTH), NUMERICAL MECHANICS APPLICATIONS INTERNATIONAL SA (NUMECA), IBM IRELAND LIMITED (IBM).

The content of this document is the result of extensive discussions within the AllScale Consortium as a whole.

More information

Public AllScale reports and other information pertaining to the project are available through the AllScale public Web site under http://www.allscale.eu.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	29/02/16	First draft	Herbert Jordan
0.2	24/03/16	Integration of review feedback	Herbert Jordan
1.0	24/03/16	Final version	Herbert Jordan

Table of Contents

Executive Summary 6
1 Introduction 6
2 Overview 6
2.1 Requirements 6
2.2 Getting the Sources 7
2.3 Package Organization 7
2.4 Build and Run 7
3 API Overview 8
3.1 AllScale Core API 8
3.2 AllScale User API 8
3.3 AllScale Pilot Application Demonstrators and Prototypes 9
4 Future Work 10

Executive Summary

This document describes the status and structure of the first AllScale API implementation deliverable (D3.1). It outlines the organization of the development process, instructions on how to obtain the sources, build the project, and execute demonstrators and test cases, as well as the internal file structure of the software project. It concludes by a summary of future work.

1 Introduction

The deliverable D3.1 is a source code deliverable, providing implementations of the AllScale API. This document provides an overview on the internal structure and organization of the sources, as well as instructions on how to obtain and build them.

The AllScale API is implemented in two phases:

- Early prototypes of API constructs are developed to explore the design space, clarify requirements, and to investigate interface design options and usability issues. Those are developed in the context of an isolated prototype software project (*allscale-api-prototype*).
- The results obtained by working on the prototypes are then migrated to the actual AllScale API development project, covering the full functionality required for the codes to be processed by the AllScale toolchain.

The prototype stage enabled us to start investigating the design of the User API and the structure and requirements of the pilot applications long before the interfaces towards underlying components, in particular the AllScale Compiler and the AllScale runtime system, have been established as a first draft. Also, relaxed requirements on the implementations lead to faster development cycles.

Within this report, however, the structure and properties of the second stage, the actual AllScale API development project (D3.1), are covered.

2 Overview

The AllScale API package provides the interfaces, reference implementations, use case demonstrations, and unit tests for the AllScale Core API, and the AllScale User API.

2.1 Requirements

The sources of this package are using features specified by the C++14 standard. Thus, a compliant C++ compiler (e.g. GCC >= 4.9) is required. For the build and testing process, cmake (Version >=2.6) is needed. Furthermore, for the build process an internet connection to download the sources for the utilized unit-testing solution (googletest) is required. No additional dependencies to third-party components are instated.

2.2 Getting the Sources

The sources are provided by the project's source-code hosting infrastructure[1] and can be obtained utilizing git. To obtain the sources, the command

```
git clone git@goedis.dps.uibk.ac.at:herbert.jordan/allscale_api.git
```

can be used[2]. This will obtain the latest version of the sources.

2.3 Package Organization

The directory and file structure of the package are organized as follows:

- <project root>
  - code – all headers, sources, unit tests, and build scripts
    - include – header files
      - allscale/api/core – core API headers
      - allscale/api/user – user API headers
      - allscale/utils – general utilities
    - src – sources
    - test – unit tests and prototype demonstrators
      - core – unit tests for the core API
      - user – unit test for the user API
      - demos – pilot application demonstrators
  - CMakeLists.txt – cmake build script
  - README.md – short summary and usage hints
  - createDebug.sh / createRelease.sh – project initialization scripts

2.4 Build and Run

Change to your local <project root> directory and create a debug build as follows:

```
mkdir build_debug
cd build_debug
. ../createDebug.sh
make –j8
```

The script *createDebug.sh* uses *cmake* to create a build environment. It uses g++ as its default compiler and may be customized for other toolchains.

To run the test cases and prototypes within the project, run

```
make test ARGS=-j8
```

within the build directory or execute individual unit tests or prototypes by running the corresponding *ut_** executable directly.

[1] At the moment of writing this document, the active source-code repository is still the temporary gitlab system provided by UIBK (https://goedis.dps.uibk.ac.at)

[2] Due to confidentiality constraints, the access to the source-code repository is restricted to a selected set of AllScale consortium members. Access to reviewers can be provided upon request.

To build and run a release build, run the commands

```
mkdir build_release
cd build_release
. ../createRelease.sh
make -j8
make test ARGS=-j8
```

within your local project root directory.

3 API Overview

This section briefly outlines the AllScale API implementation covered by Deliverable D3.1. For a detailed specification of the API we refer to Deliverables D2.5 – AllScale API Specification (a) and D2.6 – AllScale API Specification (b) respectively.

3.1 AllScale Core API

The AllScale Core API covers three essential components:

- a higher-order function creating parallel recursive operations (prec)
- futures for synchronizing parallel recursive tasks
- an abstract *Data Item* interface for data objects being distributed and managed by the AllScale runtime system

The following table lists those central elements of the AllScale Core API and the corresponding files relative to the *<project root>/code/include/allscale/core* directory:

Construct	**Header**	**Reference Implementation**
prec	prec.h	prec.h
decomposable futures	future.h	impl/reference/runtime.h
data items	data.h	data.h

3.2 AllScale User API

The following table lists the essential elements of the AllScale User API and the corresponding files relative to the *<project root>/code/include/allscale/user* directory:

Construct	**Header**
parallel loop	operator/pfor.h
stencil	operator/stencil.h

adaptive grid stencil*	operator/adaptive_grid_stencil.h
vector	data/vector.h
distributed key/value map	data/map.h
n-D dynamic grid	data/grid.h
n-D adaptive grid*	data/adaptive_grid.h
unstructured multi-grid mesh*	data/mesh.h
geometry model*	data/geometry.h

Note: prototype implementations of the User API constructs have been developed in the context of the API prototype project to be obtained from

```
git@goedis.dps.uibk.ac.at:herbert.jordan/allscale_api-prototype.git
```

Currently (mid-March 2016), those implementations are gradually migrated to the AllScale API project covered by this deliverable. Entries in the table above marked by a * are still in the process of migration.

3.3 AllScale Pilot Application Demonstrators and Prototypes

The following table lists the demonstrator and prototype implementations of the AllScale Pilot Applications utilized to validate and test the AllScale API. Paths are relative to the *<project root>/code/test/demos* directory:

Pilot Application	**Demonstrator / Prototype**
AMDADOS	amdados_demo.cc
iPIC3D	ipic3d_demo.cc
Fine/Open	fine_open_demo.cc

Note: as for the User API, the demonstrators have been developed in the context of the API prototype project obtainable from

```
git@goedis.dps.uibk.ac.at:herbert.jordan/allscale_api-prototype.git
```

Currently (mid-March 2016), the demonstrators are in the process of being migrated to the locations stated by the table above.

4 Future Work

Currently, the development of the AllScale API is focusing on porting implementations of various constructs from the relaxed prototype stage to the fully-functional development project. The main targets here are the porting of the adaptive grid and mesh data structures as well as the associated operators. This phase will be concluded by porting the pilot code demonstrators to the development project.

Future steps will target the provisioning of improved data structures for the pilot codes, including improved implementations of collections (e.g. particle list for iPIC3D), memory efficient implementations of the adaptive grid (AMDADOS) and efficiently distributable, cache friendly mesh data structures (Fine/Open).

At the same time, the results of the development of the runtime interface, in particular its requirements on data objects to be managed by the system, will be integrated into the code base.