H2020 FETHPC-1-2014

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

Project Number 671603

D6.9 – Installation, integration and deployment of the AllScale environment and pilot applications (b)

WP6: Integration testing and pilot applications

Version: 2.0
Author(s): Fearghal O'Donncha
Date: 29/05/2018
D6.9 – Installation, integration deployment AllScale environment & applications (b)

**Due date:** PM32

**Submission date:**

**Project start date:** 01/10/2015

**Project duration:** 36 months

**Deliverable lead organization** IBM

**Version:** 2.0

**Status** Final

**Author(s):** Fearghal O’Donncha

**Reviewer(s)** Alex Hirsch, Stephane Monte

**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.
D6.9 – Installation, integration deployment AllScale environment & applications (b)

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 (UIBK), 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

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comments, Changes, Status</th>
<th>Authors, contributors, reviewers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>05-01-2017</td>
<td>First draft</td>
<td>Fearghal O'Donncha, Emanuele Ragnoli</td>
</tr>
<tr>
<td>0.2</td>
<td>06-01-2017</td>
<td>Comments from Kiril Dichev</td>
<td>Kiril Dichev, Fearghal O'Donncha</td>
</tr>
<tr>
<td>0.3</td>
<td>09-01-2017</td>
<td>Comments from Erwin Laure</td>
<td>Erwin Laure, Fearghal O'Donncha</td>
</tr>
<tr>
<td>0.4</td>
<td>17-01-2017</td>
<td>Comments from Benoit Leonard and Peter Thoman</td>
<td>Benoit Leonard, Peter Thoman, Fearghal O'Donncha</td>
</tr>
<tr>
<td>1.0</td>
<td>17-01-17</td>
<td>Final version of D6.8(a)</td>
<td>Fearghal O'Donncha</td>
</tr>
<tr>
<td>1.1</td>
<td>03-05-18</td>
<td>First version of D6.9(b)</td>
<td>Fearghal O'Donncha</td>
</tr>
<tr>
<td>1.2</td>
<td>25-05-18</td>
<td>Reviewers comments Alex Hirsch</td>
<td>Fearghal O'Donncha</td>
</tr>
<tr>
<td>1.3</td>
<td>25-05-18</td>
<td>Reviewers comments Stephane Monte</td>
<td>Fearghal O'Donncha</td>
</tr>
<tr>
<td>2.0</td>
<td>29-05-18</td>
<td>Final version</td>
<td>Fearghal O'Donncha</td>
</tr>
</tbody>
</table>
## Table of Contents

Executive Summary .............................................................................................................. 6

1 Introduction ....................................................................................................................... 7

2 Infrastructure ..................................................................................................................... 7

2.1 In-house infrastructure with controlled access .............................................................. 7

2.1.1 Continuous Integration ............................................................................................ 8

2.1.2 Source Control ............................................................................................................. 9

2.2 Public-facing repository ................................................................................................. 11

3 Working with the AllScale environment ........................................................................... 11

3.1 Building the AllScale environment ................................................................................ 11

3.2 Testing and Evaluation of AllScale Environment ......................................................... 12

3.3 Pilot Application Integration ......................................................................................... 13

3.3.1 AMDADOS ................................................................................................................ 13

3.3.2 IPIC3D ........................................................................................................................ 13

3.3.3 Fine/Open .................................................................................................................... 14

3.4 Deployment on HPC architecture ................................................................................ 15

4 Summary .......................................................................................................................... 15

Appendix .................................................................................................................................. 15
Executive Summary

This document describes the status and structure of the installation, integration, and deployment of the AllScale environment and pilot applications (D6.9). This document builds on D6.8 to provide a complete overview of the AllScale software environment and pilot applications, obtaining and building the AllScale toolchain, and integrating with application code using the pilots as case study. We discuss the details of a continuous integration (CI) environment hosted on one of the consortium partners (QUB) infrastructure. The CI environments consists of multiple compute nodes for functional testing, a GitLab server for source control and a server running Jenkins for continuous integration tests and source code fidelity. The document presents the different components of the AllScale codebase and how we implemented the CI environment for the different modules.

The main differences between this deliverable and D6.8 are:

- Provides a complete overview of how to obtain and build the AllScale toolchain.
- Provide a Dockerfile to allow users to quickly configure their environment and install all AllScale dependencies.
- Describe the process to build and run the pilot applications with some benchmark configurations to allow interested users to easily run some large-scale test cases.
- Describe the updated software repositories used by the project.
- Assess the AllScale environment with regards to accessibility and ease-of-use from the application developers' perspective.
1 Introduction

To enable collaboration and feedback among consortium partners’ requirements and needs (AllScale environment development, testing and validation and integration with the three pilot applications), were collected and agreed among the partners between M3 and M6 of the project. In general terms those consisted of:

1. A single shared repository that allows build automation, testing, and deployment with defined security based on industry standards.
2. A software version control system to maintain current and historical versions of files such as source code, and documentation.
3. A functional testing environment with computational resources to ensure that minimal performance and computational benchmarking requirements are met.
4. All of the above to be based on open source products.

Based on above requirements QUB offered to host the infrastructure on one of their servers at no cost to the project. Details of the infrastructure are in the following sections.

As the project progressed and the technological maturity of software assets reached alpha levels of production, key components of the AllScale toolchain were move to a public GitHub to promote usage, feedback and contributions from the scientific community.

2 Infrastructure

First we introduce the in-house continuous integration (CI) and version control infrastructure that we used for development during the project. Later sections describe the public facing GitHub repositories that we use.

2.1 In-house infrastructure with controlled access

The continuous integration and functional testing infrastructure are hosted by QUB in the Institute for Electronics Communications & Information Technology (ECIT). The infrastructure was built on open-source software hosted on a single server freely provided by QUB. VPN access is provided to all consortium members in line with system administration security requirements. The software management system consists of two distinct systems:

- A source code repository server to store and exchange the code of AllScale components.
- A continuous integration system for the ongoing testing of modifications to the AllScale components.

Table 1 presents the identified requirements of the repository server and CI system.
Table 1: Requirements for source control and CI system

<table>
<thead>
<tr>
<th>Source code repository server</th>
<th>Continuous integration (CI) system</th>
</tr>
</thead>
<tbody>
<tr>
<td>• very low performance and memory requirements (e.g. one node, single core, 2 GB RAM)</td>
<td>• small distributed memory cluster (e.g. 4 nodes, each multi-core and multi-processor, 32 GB RAM, ideally equipped with GPUs)</td>
</tr>
<tr>
<td>• allows source code exchange</td>
<td>• used for functional testing only, not for performance or scalability testing</td>
</tr>
<tr>
<td>• needs to be accessible to all AllScale consortium members for pulling/pushing changes</td>
<td>• needs to be able to pull changes from the source code repository server and publish the results e.g. via HTTP</td>
</tr>
<tr>
<td></td>
<td>• runs all AllScale test cases whenever a new version of the infrastructure or the project package is pushed to the source repository</td>
</tr>
</tbody>
</table>

The infrastructure consists of one ESX R520 running 2.40 GHz Intel(R) Xeon(R) CPU E5-2407 v2 with 4 cores. The server provides 192GB RAM and 2 x 1TB SATA (Serial Advanced Technology Attachment) disk storage. The server hosts eight virtual machines (VMs) each consisting of one virtual node hosting Ubuntu version 14.04 operating system (OS). The virtual nodes are 4 test or compute nodes (for functional testing), one hosting GitLab server, one hosting Jenkins server, and a node each dedicated to the lightweight tasks of login management and VPN server. Each test node has 16GB RAM with 50GB of OS disk space and 100GB scratch space. The GitLab server has 2GB RAM with 50GB of OS space and 100GB scratch space. Similarly, the Jenkins server has 2GB RAM, 100GB OS space and 150GB scratch space. The system is supported by ECIT system administration with a Request Tracker ticketing system.

2.1.1 Continuous Integration

The continuous integration system structure is composed as follows:
D6.9 – Installation, integration deployment AllScale environment & applications (b)

- Four nodes dedicated to running functional tests using the AllScale environment
- An additional node for compilation and management running Jenkins server.
- The Manager/Compilation node must be able to access all source code repositories and hence have access to all AllScale components (both that contained locally in GitLab and on the internet on GitHub).
- No direct access to this system is required except for system setup and administration. Read access is provided to all developers with administration rights restricted to system administrators.

Figure 2 presents an example of a typical CI loop performed by the system:

![CI Loop Diagram](image)

**2.1.2 Source Control**

To maintain the software repositories for the AllScale project a single instance of a GitLab installation on the source repository server is used. This server is hosted at the QUB facilities and is used to host codes with proprietary components. Within this installation, several projects are maintained, each covering a specific component of the project and managed by the corresponding work package/task leader. Secondary to this, some components were developed outside the AllScale project and reside on GitHub repositories accessible to the CI system.
The source code repository structure consists of several different components:

1) Software components (rectangles in the figure) are Git repositories, dependencies between them are implemented as GitHub submodules.

2) Pre-existing components are available in the public domain (e.g. GitHub) and will remain so; AllScale components, aggregation repositories and application repositories are available to all AllScale consortium members either via GitLab repositories for codes with proprietary aspects or via Github for codes with open access policy.

3) Aggregation packages will combine other components and add little-to-nothing new content on their own. Their main purpose is to allow a single, consistent version of the AllScale environment and the entire project.

4) The pilot applications only reference the “AllScale Environment” package, facilitating referencing specific versions and ensuring a consistent and stable system.

5) All components, including the pilot applications, are referenced by a common top-level package (“AllScale Project” in the figure), and must be available to all members participating in the development. If any component or application is not available in the proper version publicly or on the protected repository server, its compatibility with other AllScale components cannot be ensured by individual developers.

The components included in Figure 3 consist of:

- The **green packages** (parec, Insieme, HPX) will be obtained from external sources and are expected to be maintained and tested externally.
D6.9 – Installation, integration deployment AllScale environment & applications

- The **blue packages** are contributions developed for the AllScale infrastructure and its evaluation. Those will be managed, developed and tested by the corresponding work-package/task leaders.
- The **orange packages** are managed by task T6.3 for the integration of the AllScale components and to provide a consistent view on the infrastructure to the pilots, CI system and externals.
- The **red packages** are managed, developed and tested by the pilot groups and combined and tested with the other AllScale components on the CI system (by the AllScale project package).

2.2 Public-facing repository

As the AllScale toolchain reached a suitable level of maturity it was moved to a public repo to enable public interaction and usage. Specifically, the AllScale Compiler, Runtime and API are hosted on GitHub at: https://github.com/allscale. It enables simpler access (public internet) and improves dissemination efforts by allowing distribution of the code to pertinent institutions, forums and people.

3 Working with the AllScale environment

3.1 Building the AllScale environment

This section reports on the process of installing the AllScale environment and compiling one’s own code using the toolchain – in this case the pilot applications are used as demonstrations.

For a typical user's perspective there are two development steps to use the AllScale environment. The first involves obtaining the AllScale SDK which provides the API and is used to develop the application code and express parallelism according to AllScale constructs. The process for obtaining, building and testing the AllScale SDK and API is relatively straightforward. First the user obtains and initializes the AllScale SDK using the provided:

```bash
$ git clone https://github.com/allscale/allscale_sdk demo_app
$ cd demo_app
$ scripts/setup/run demo_app
```

The next step is to build the project by implementing the following steps:

```bash
$ mkdir build
$ cd build
$ cmake ../code
$ make -j8
$ ctest -j8
```

Once built and tested, the user can develop/port their code leveraging API data structures and parallelism and compile using the standard C++ toolchain. This allows the user to rapidly build/test their code but it does not provide the full capabilities of the AllScale environment and runtime – this demand one uses the AllScale compiler rather than the standard C++ toolchain. However, it provides a
D6.9 – Installation, integration deployment AllScale environment & applications

valuable approach for rapid development and prototyping with low technological
demands (only requires the AllScale API and standard C++ compilers). More
details on building code within the AllScale API is provided in the pilot application
sections.

The next stage in the AllScale development process is obtaining and building the
AllScale compiler. This provides to the user the full capabilities of the AllScale
environment. A set of scripts are provided to build and install the AllScale
compiler on Debian/Ubuntu type operating systems. On these systems the process
to obtain and build the AllScale environment is exceedingly simple. Following the
instructions of the README (https://github.com/allscale/allscale_compiler):

```
$ git clone --recursive https://github.com/allscale/allscale_compiler
$ cd allscale_compiler
$ ./QUICKSTART  # prompts for sudo
```

The AllScale compiler installation script also builds the runtime (from
https://github.com/allscale/allscale_runtime) and all other AllScale. The next
section presents a test installation and integration of the AllScale environment at
IBM facilities to evaluate ease of use and accessibility of the AllScale environment
to application developers.

### 3.2 Testing and Evaluation of AllScale Environment

Testing and evaluation was done on an X86_64 machine as the QUICKSTART script
provided only supports X86_64. All tests were done inside a Docker container
running Debian 9.4. Docker is an open-source platform for container-based
virtualization and the management of Linux containers. Docker containers are
extremely lightweight virtual machines that allow code to be run in isolation from
other containers. The Dockerfile used for this deliverable is placed in appendix at
the end of this document.

Installation of the AllScale compiler (and runtime) followed the process
described above: git clone and execute the QUICKSTART script. Testing on the Docker image,
resulted in zero issues or errors as shown in Figure 4.

```
root@eb3787e7d9f5:/work/allscale_compilers$ tail allscale_compile.log
234/239 Test #232: ut_compiler_core_data_item_conversion_test .................. Passed 32.46 sec
235/239 Test #234: ut_compiler_core_distributed_memory_checks_test ........ Passed 35.34 sec
236/239 Test #235: ut_compiler_frontend_allscale_fe_conversion_test .......... Passed 34.19 sec
237/239 Test #236: ut_compiler_analysis_input_test ............................ Passed 138.73 sec
238/239 Test #228: ut_compiler_backend_allscale_code_snippets_test .......... Passed 459.06 sec
239/239 Test #146: ut_driver_integration_overall_integration_test .......... Passed 593.04 sec
100% tests passed, 0 tests failed out of 239
```

**Figure 4: Successful installation of the AllScale environment**

Installing the AllScale API went along similar lines. Cloning from the Github
repository and following the instructions from the README, the AllScale API was
installed in a matter of minutes. During this project, the three pilot applications
were ported to this API. As a guide to future users, we now provide, for each of the
pilot applications, an overview of the code structure, compilation process and
execution using the AllScale toolchain.
3.3 Pilot Application Integration

As a demonstration of AllScale accessibility and ease-of-use, this section describes the integration and running of pilot applications within the AllScale environment. It aims to serve as a guide on typical build and execution structures for applications within AllScale.

3.3.1 AMDADOS

By adopting the same file structure as the API (recommended), the procedure to build the AMDADOS application is equivalent to building the API using a cmake, make and ctest pipeline. By default, the application compiles using the standard toolchain. To run using the AllScale toolchain, one must compile with the appropriate compiler. This is done via a simple cmake flag, namely ‘USE_ALLSCALECC=ON’ and involves no other change to the build process. The application can be run in parallel using either the standard or AllScale toolchain (this is specified only at compile time). The process is as simple as launching the application while specifying the number of parallel threads, i.e. to run 6 threads:

```bash
$ export NUM_WORKERS=6
$ ./amdados --config amados.conf
```

Application specific configurations are specified in the amados.conf file. The most relevant parameters are the number of subdomains (controlling the total size of the problem) and the number of sensors with observations (dictating data assimilation). To facilitate testing and experimentation, a Python script is included to generate observations based on user defined configurations (from the amados.conf file). The full process to run the application (from the root directory) is then:

```bash
$ python3 python/GenerateObservation.py --config amados.conf
$ /build/app/amdados --config amados.conf
```

3.3.2 iPIC3D

As AMDADOS, iPIC3D – implicit particle-in-cell code for space weather simulations – adopts the same file structure as the API suggested. Building the pilot is quite straightforward by following the user guide: cmake, make, and ctest (to check correctness). Correctness is verified automatically by collecting the conserved aggregated data – such as total energy and total momentum – as well as snapshots of fields’ values defined on the grid and particles distribution among cells. By default, the AllScale iPIC3D pilot uses the AllScale Standard toolchain. This can be changed to the AllScale toolchain during the build as in the previous section.

Before launching the pilot, one should create a directory with name ‘data’ to store the outputs. The application can be launched by the following command

```bash
$ /build/app/ipic3d /inputs/tiny.inp
```

All the inputs are provided in the inputs sub-directory and are focused on the Earth’s radiation belts simulation. This test case simulates the formation of the Earth’s radiation belt: it commences with uniformly distributed particles in space with the Maxwellian distribution in velocity and, over the course of the simulation; it converges to the stage with the majority of particles being trapped in the
radiation belts. Thus, the input files specify, in particular, the grid size, the particles distribution in three directions cell-wise, and the other auxiliary simulation parameters required for the proper initialization of the simulation.

At the end of the simulation, the output files and the aggregated conserved quantities would be stored in the ‘data’ directory.

3.3.3 Fine/Open
As for other pilot applications, Fine/Open adopts the same file structure as the API suggested. The project is initialised using the SDK and built following the steps described in Section 3.1. The solver computes the evolution of the flow and return the solution in .det files which then need to be post-processed. The FINE/Open pilot application is a multi-grid compressible steady and unsteady Navier & Stokes solver including one turbulence model (Spalart-Allmaras) dedicated to the aero-external applications.

In order to run a computation, using a configuration file (.run file):

First we pre-process the mesh (the only mesh format compatible is the AS_Full format produced by Numeca for the consortium). To run the pre-processing step:

\[
\text{preprocessor config.run}
\]

Then you can run the solver:

\[
\text{solver config.run}
\]

At last to visualize the results with paraview, you need to post-process the results. The post-processor will generate a vtk file containing the mesh and the solution.

To run the post-processing step:

\[
\text{post-processor config.run}
\]

Figure 5 presents the entire structure to pre-process data inputs, run the model and post-process the data into required format for human interaction.

![Figure 5: Process to configure, build and run the Fine/Open application](image-url)
3.4 Deployment on HPC architecture

The AllScale toolchain has been installed on in-house HPC architecture by all project partners. Generally, installation was done by one of the AllScale team who were familiar with the structure, dependencies and installation processes of the software.

A notable achievement and useful external assessment however, was the successful installation on KTH infrastructure (https://www.pdc.kth.se/hpc-services/computing-systems/tegner-1.737437) by the local system administrators. The system was running CentOS v7.4 and gcc7.2 was used for compilation. Due to the rpm based package manager system the QUICKSTART script provided was not functional for this OS (focuses on Debian/Ubuntu). Dependencies (that were not already present with appropriate version) had to be installed via the installer scripts provided with the INSIEME and AllScale compiler. To ensure appropriate versions, these are installed from source by downloading tar files rather than from package manager. The process was noted as being relatively cumbersome, which is somewhat expected considering the number of dependencies that the AllScale compiler and runtime dictate. As noted previously, the only supported operating system is Debian type and installing on other systems are not supported by the AllScale developers.

4 Summary

This document presents details, from the perspective of pilot application developers, of installing and building the AllScale environment and integrating with the respective pilot applications. Within a standard Debian environment installation proceeded with no manual intervention. Details on obtaining and installing the AllScale pilot applications are also presented; the authors believe that these present valuable assets to the user to get familiar with developing and building within the toolchain.

The Installation, Integration and Deployment AllScale Environment is capable of fostering testing, development and validation among the consortium partners. All the AllScale pilots are deployed and being benchmarked and tested within it with the AllScale software stack. Results on pilot application performance using the toolchain are presented in M31 deliverables D6.3, 6.5 and 6.7 representing the iPIC3D, Fine/Open and AMDADOS applications respectively.

Appendix

Docker file used for all experiments described in this deliverable:
To run this docker image, simply copy the below text to file named 'Dockerfile' and from same directory run following two commands:

```
$ docker build --tag dev-allscale:compiler-env . -f Dockerfile
$ docker run -it IMAGE_NUMBER
```

To find image number, simply type 'docker image'.

```bash
# Building:
# docker build --tag dev-allscale:compiler-env . -f Dockerfile
```
# Running:
# docker run -it IMAGE_NUMBER
# Entering running container:
# docker exec -it CONTAINER_NUMBER /bin/bash
# Extracting for sharing on another machine:
# docker save dev-allscale:compiler-env >
# allscale_compiler.tar
# Loading on another machine:
# docker load < allscale_compiler.tar
#
# FROM debian:9.4

LABEL maintainer="Fearghal O'Donncha, feardonn@ie.ibm.com"

ENV LC_ALL=

# System package.
RUN apt-get update && apt-get -y upgrade && 
   apt-get install -y --no-install-recommends 
   build-essential gdb gfortran cmake make automake 
   unzip bzip2 curl wget rsync tmux htop 
   libfreetype6-dev libpng12.* libzmq3-dev 
   libjpeg-dev libtiff[0-9]*-dev librsvg2-dev 
   pkg-config software-properties-common 
   libcurl.*-openssl-dev libpcre++-dev libxml2-dev 
   git mc vim nano make cmake gdb 
   openssh-client openssh-server 
   binutils binutils-dev sshfs exuberant-ctags valgrind 
   libblas-dev liblapack-dev liblapacke-dev 
   libarpack2-dev libarpack2++-dev libopenblas-dev 
   libatlas-base-dev libsuperlu.*-dev libopenblas-dev 
   && 
   apt-get clean && apt-get autoremove && 
   rm -rf /var/lib/apt/lists/* 
   && 
   ssh-keygen -t rsa -f ${HOME}/.ssh/id_rsa -q -P "" 

# Working directory.
WORKDIR /root/work

# Execute this command on start-up.
CMD ["/bin/bash"]