Deliverable 4.1: Report and software on design of tools and required actions to support performance tools for multiscale

Due Date	September 2016
Delivery	Month
Lead Partner	Allinea Software Ltd.
Dissemination Level	Confidential
Status	Final
Approved	Executive Board yes/no
Version	V1.3

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 671564.

DOCUMENT INFO

Date and version number	Author	Comments
18.07.2016 v0.1	Keeran Brabazon	Outline of the report with headings of main sections
08.08.2016 v0.2	Keeran Brabazon	Draft content for Allinea checking
12.08.2016 v0.3	Keeran Brabazon	Updates from Allinea feedback
18.08.2016 v1.0	Keeran Brabazon	Draft for Work Package 4 members
05.09.2016 v1.1	Keeran Brabazon	Update after Work Package 4 review
06.09.2016 v1.2	Keeran Brabazon	Update after PSNC comments
26.09.2016 v1.3	Keeran Brabazon	Final draft

CONTRIBUTORS

Contributor	Role
Dirk Schubert	WP4 Contributor, WP4 Leader
Keeran Brabazon	WP4 Contributor
Stephan Hachinger	WP6 Contributor
Vytautas Jancauskas	WP6 Contributor, WP6 Leader
Tomasz Piontek	WP5 Contributor, WP5 Leader
Piotr Kopta	WP5 Contributor
Saad Alowayyed	WP3 Contributor

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

1 Executive summary

This document summarises the work done so far in the design of the parallel debugging and performance analysis tools to be used in ComPat. As per the Description of Actions (DoA), work so far has focussed on the gathering of requirements for the tools from project partners, as well as their high level design (see Tasks 4.1-3) such that they can be used in an effective manner in the development of multiscale simulations. The tools used as part of ComPat are provided by Allinea Software Ltd., and for the remainder of the report any mention of 'tools' implicitly refers to Allinea tools. The work performed so far has been on target with the timeline in the DoA. In particular, this document serves as proof that "Milestone 4: Completion application and pattern / framework design" has been completed.

Task 4.1 is responsible for the gathering of requirements from users and the design of an architecture for debugging and profiling multiscale applications. The requirements gathering phase has been completed, as well as the initial architecture of solutions. The requirements gathered are discussed in this document alongside a description of how the current architecture supports the development of multiscale applications, and how extensions to the current architecture will support software development in a multiscale environment. Mention is made how the tools have already been used as part of the project.

Tasks 4.2 and 4.3 (Energy and Performance Profiling and Expert Advice) have begun and are in an investigative phase in order to establish the feasibility of proposed solutions and strengthen the design of product extensions.

The tools are mostly to be used by application developers part of Work Package 3, and the design of extensions to the tools focusses on their requirements. Application developers are to develop patterns outlined in Work Package 2. Discussion with Work Package 2 members has helped to identify general application workflows that need to be supported by the tools. This includes debugging and profiling of single scale models, scale bridging components, as well as entire multiscale simulations. This will support performance and energy analyses of the ComPat applications and will be used to verify performance prediction models developed.

Middleware services developed in Work Package 5 are to be used to enable communication between component applications in a multiscale simulation. Allinea tools are to be integrated with the middleware services as part of the project, so close collaboration with Work Package 5 will be maintained. Requirements have also been gathered from Work Package 6, which defines the platforms and systems on which Allinea tools need to run, as well as software which the tools depend on.

The rest of the report is structured as follows. Chapters 2-4 introduce the tools to be used in ComPat. This starts with a discussion of how the tools can be used in their current state, and describes extensions necessary to meet the requirements gathered. Chapter 5 concludes the report and provides a summary of the progress and achievements obtained so far.

2 Debugging with Allinea DDT

In this chapter the parallel debugger Allinea DDT (from now on DDT) is introduced. We start with a description of the current design and functionality of the debugger. Focus is kept on the application of the debugger in a multiscale environment, where support is required for simultaneously debugging multiple applications as part of a multiscale simulation. Areas are identified where functionality of the debugger can be extended to give further support to debugging multiscale simulations. The new functionality is described alongside existing functionality where appropriate. New functionality not related to existing functionality is outlined in Section 2.2.

2.1 Current State

DDT is a highly scalable parallel debugger. It allows for high performance codes written in C/C++ and FORTRAN running an MPI and/or OpenMP and/or CUDA model of execution to be debugged. For a detailed description of how to run and use the tools please see the Allinea DDT User Guide [1]. DDT is designed to be easy to use, without requiring recompilation or linking of code (assuming that the debug flag has been set, and optimisations have been turned off) on all current HPC systems where dynamic linking is available.

A debugging session can be interactive or offline. In order to start a debugging session it is simply required to prefix a small amount of text to the start of the command line used to run an application. For example, if a parallel application is launched as follows:

```
$ mpirun -n 256 ./parallel_program
```

then in order to start an interactive debugging session the command line would change to:

```
$ ddt mpirun -n 256 ./parallel_program
```

A list of commands used to start Allinea tools can be found in Table 1 An interactive debugging session requires user interaction and the user has complete control over the processes used in a multi-process application. An example of an interactive debugging session is given in Figure 1. The program being debugged is using approximately 25 thousand cores. Due to the scalable nature of the product, both the debugger and the underlying application remain responsive. This makes the tool very useful in debugging large applications run on a single HPC facility.

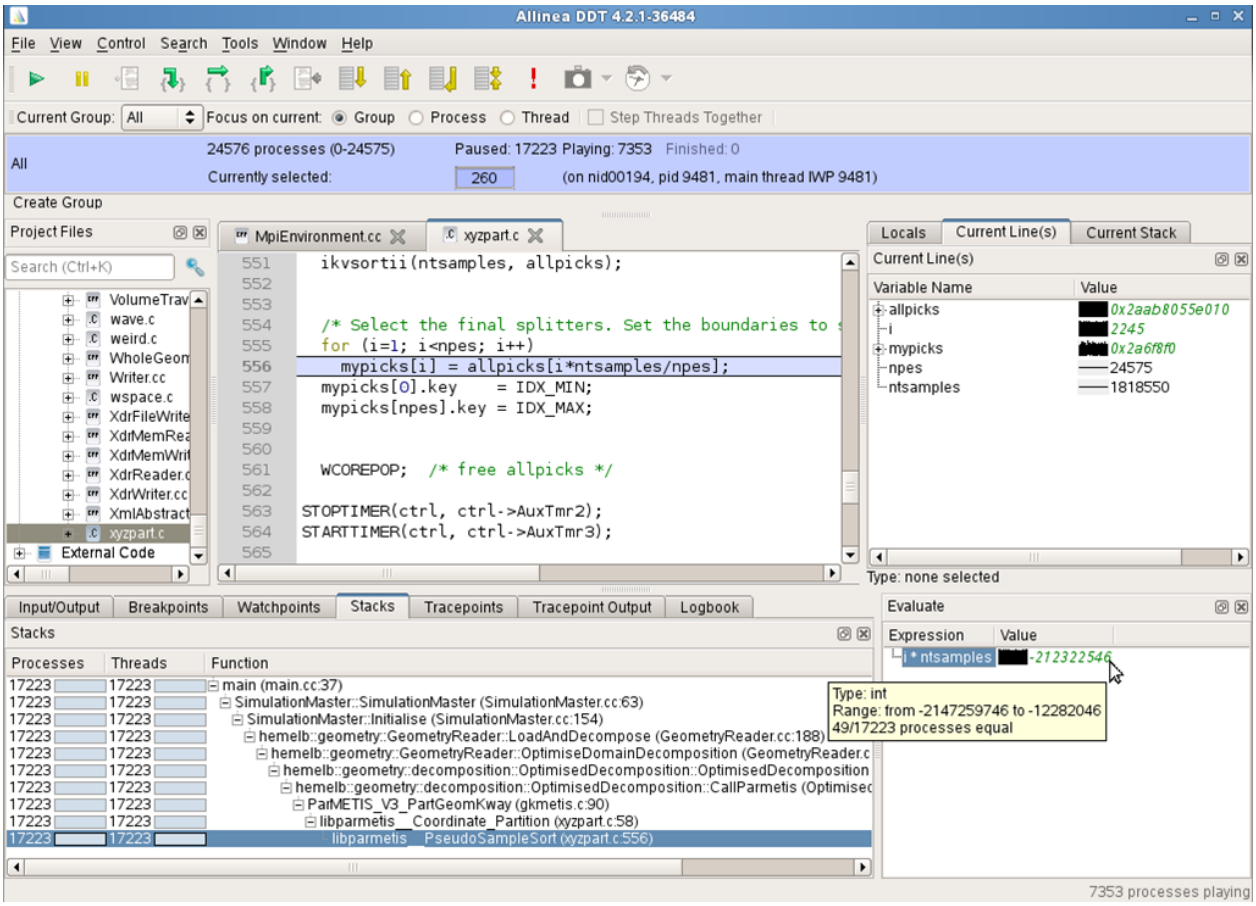
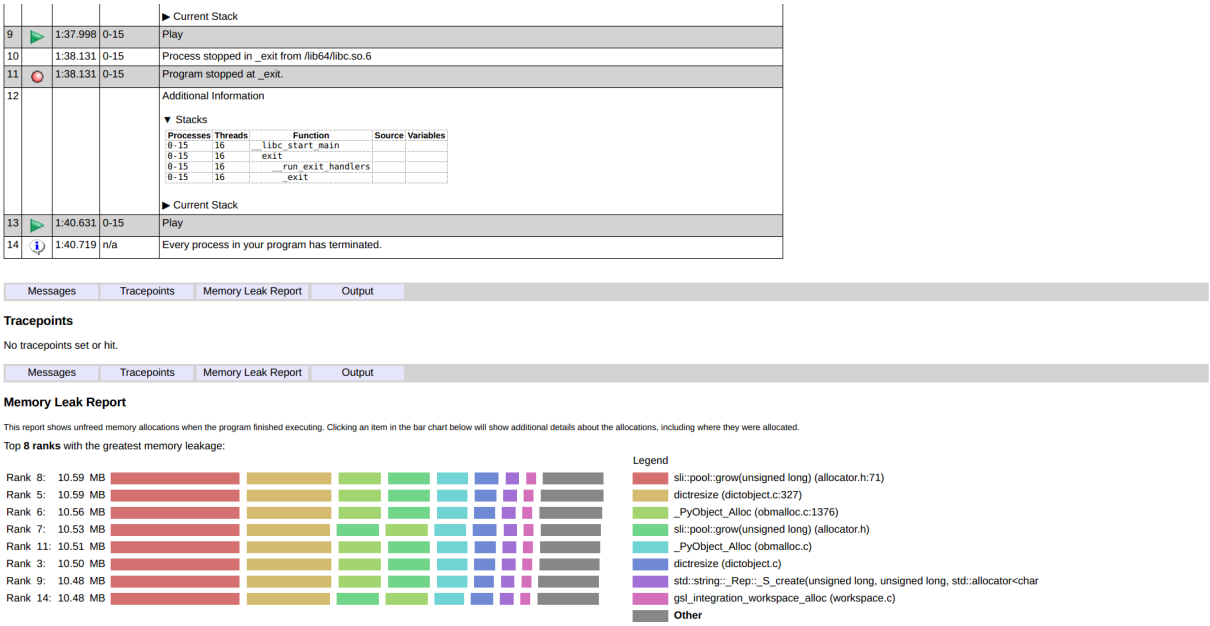


Figure 1: An interactive DDT session. This is only part of the rich interface. It is possible to see in the panes (left to right, top to bottom): File browser for files; Code viewer for source code being debugged; Values of variables on the current line / stack; Stack frames for processes; Evaluation of native code expressions.

An offline debugging session can be used to record debug information, such as stack traces at program crashes, when tracepoints and breakpoints are hit, or common memory errors such as leaks and out of bounds accesses are detected. An offline debug report is generated which can be viewed at a later date. The underlying tool is the same, but no graphical user interface is required. An example of an offline debugging report is given in Figure 2.

Debugging is often performed on a remote system, where users launch applications from a shared login or batch node, where X-forwarding is not available or slow. In this case, it is not usable to start an interactive debugging session directly on the remote system. Therefore, a feature of Allinea DDT called Reverse Connect (see Section 3.3 [1]) is required in order to control the flow of a debugged program from their own system, which is likely to be their workstation or laptop. The mechanism used for this is important for the design of integration with the ComPat middleware services developed in Work Package 5, and so is outlined in Figure 3. As can be seen in Figure 3, a GUI running on a user system communicates with a remote system via an ssh tunnel. This does not impact on the performance of the debugger from a user perspective.



				▶ Current Stack
9	▶	1:37.998	0-15	Play
10		1:38.131	0-15	Process stopped in _exit from /lib64/libc.so.6
11	●	1:38.131	0-15	Program stopped at _exit.
12				Additional Information ▼ Stacks Processes / Threads / Function / Source / Variables 0-15 16 __libc_start_main 0-15 16 exit 0-15 16 __run_exit_handlers 0-15 16 _exit ▶ Current Stack
13	▶	1:40.631	0-15	Play
14	ⓘ	1:40.719	n/a	Every process in your program has terminated.

Figure 2: An offline report contains information captured during an application run. This selection shows part of a debug report in which leaked memory is reported to the user. Memory errors, program crashes and many other common problems can be diagnosed from an offline log.

Discussion so far has focussed on debugging a single multi-process application. In a multiscale setting, in particular that presented in ComPat, the tools may already be put to good use. When interactively debugging a multiscale simulation it may be desirable to debug more than a single application (or submodel). It was considered whether this should be possible to perform from within a single DDT GUI instance. However, the richness of the information presented to the user is likely to make it difficult to separate which information is related to which application being debugged. Instead, the natural separation provided by launching multiple applications will be maintained, and the user will be responsible for debugging individual applications and submodels in separate GUIs, as depicted in Figure 4.

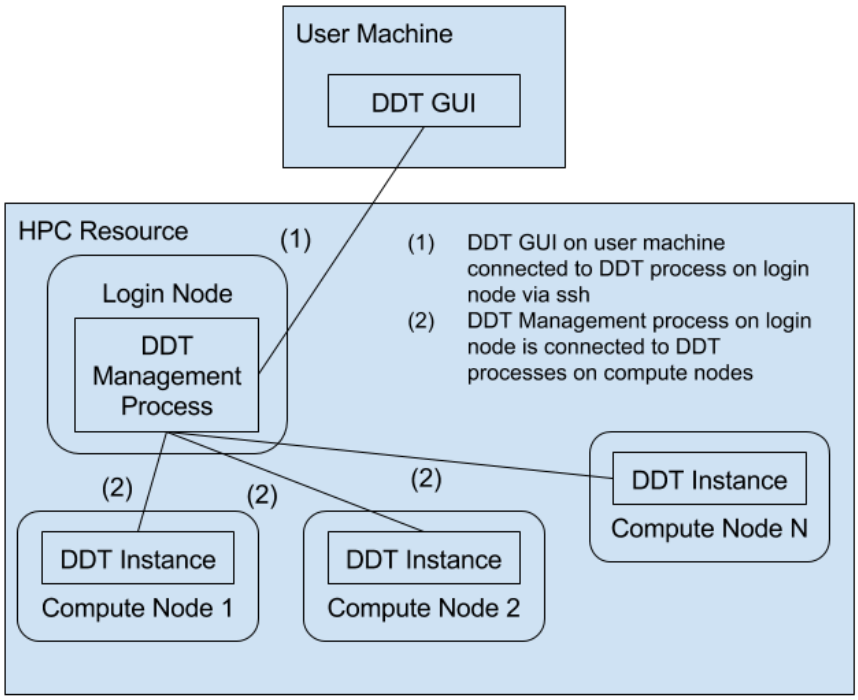


Figure 3: Structure of a remote interactive debugging session on an HPC resource. In order that the GUI on a user machine can control program flow on the HPC compute nodes, the GUI application sends (via ssh) commands to a process running on the HPC login node. Commands are forwarded to DDT instances on the compute nodes. Effects of executing the command are sent back to the user through the management process on the login node.

With respect to offline debugging, it is possible to generate offline debug reports for all applications run as part of a multiscale simulation. A user may then investigate the debug reports after their run has completed. It may also be the case that a subset of applications is desired to be debugged. If a user knows which subset of applications to generate debug reports for, they may specify this before launching their simulation. In particular, this is applicable to the Extreme Scaling and Replica Computing patterns described in Deliverable 2.1. It could be the case that there are hundreds, or thousands, of applications executed as part of a multiscale simulation that fit into these patterns. From discussions with users in Work Package 3 it is evident that it is important to be able to gather detailed information regarding a crash that occurs in one (or only a few) applications. Generating a debug report for many thousands of applications will not provide the user with salient information. Therefore, Allinea will introduce the capability of monitoring debugged runs, and only generating a debug report when some user defined condition is met. For example, only generating a report when a crash occurs, or when a tracepoint is triggered. In the absence of needing to check program internal state (such as the triggering of a tracepoint) a lightweight monitoring tool can be employed to check when an application crashes, at which point a stack trace and debug information is gathered. For an initial implementation, though, conditions will be identified in which a debug report should be generated whilst full program state is monitored. This will give a larger overhead in terms of run time, but will provide users with a solution for viewing a manageable subset of debug data.

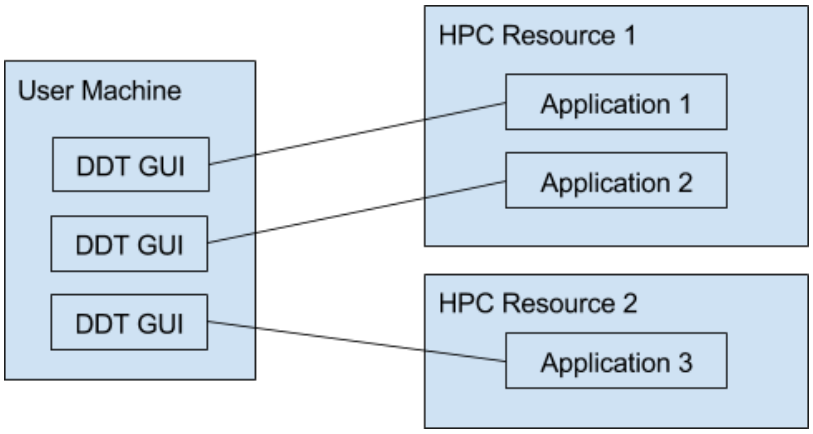


Figure 4: Each application in a multiscale simulation is controlled in a separate GUI.

2.2 Extensions

In ComPat, middleware services developed in Work Package 5 are responsible for the launching of applications on remote HPC facilities. It is therefore necessary to consider how a debugging session can be launched with the use of the ComPat middleware stack. Two tools used in ComPat – QCG [2] [3] and FabSim [4] - are able to control the flow of control of a multiscale simulation. Section 2.2.1.2 of Deliverable 5.1 states that FabSim is to be integrated into QCG, so that here we only consider integration of Allinea tools with QCG. The design for integrating QCG and DDT is applicable also to the profiling tools outlined in Chapters 3 and 4. The next section therefore presents the interaction for all Allinea tools with QCG, which is introduced in Section 2.4.1 of Deliverable 5.1. In the following discussion it is assumed that any HPC resource on which an application can be submitted by QCG has Allinea tools installed. As part of the project this is a fair assumption, and the tools are already installed on-site at PSNC, STFC and LRZ.

For integration of Allinea tools with QCG we firstly consider how Allinea tools will be launched by QCG. The commands outlined in Table 1 will need to be prepended to command line arguments for user applications. It has already been tested that debugging and profiling sessions can be started in this manner by manually inserting the commands into scripts submitted to QCG. In future work, a user may request a debugging or profiling session to begin, and QCG will inject the Allinea commands automatically. It is therefore required for Allinea to supply QCG developers with documentation for the correct syntax for starting runs. This will be of a similar form to the first column of Table 1.

As well as launching debugging and profiling runs on remote systems via QCG, it is necessary to collect data that are generated and return these to the user. Additionally, Allinea would like to store this information in a location for later access by the user. Figure 5 depicts the transfer of data between QCG nodes, HPC resources and Allinea owned resources for this case.

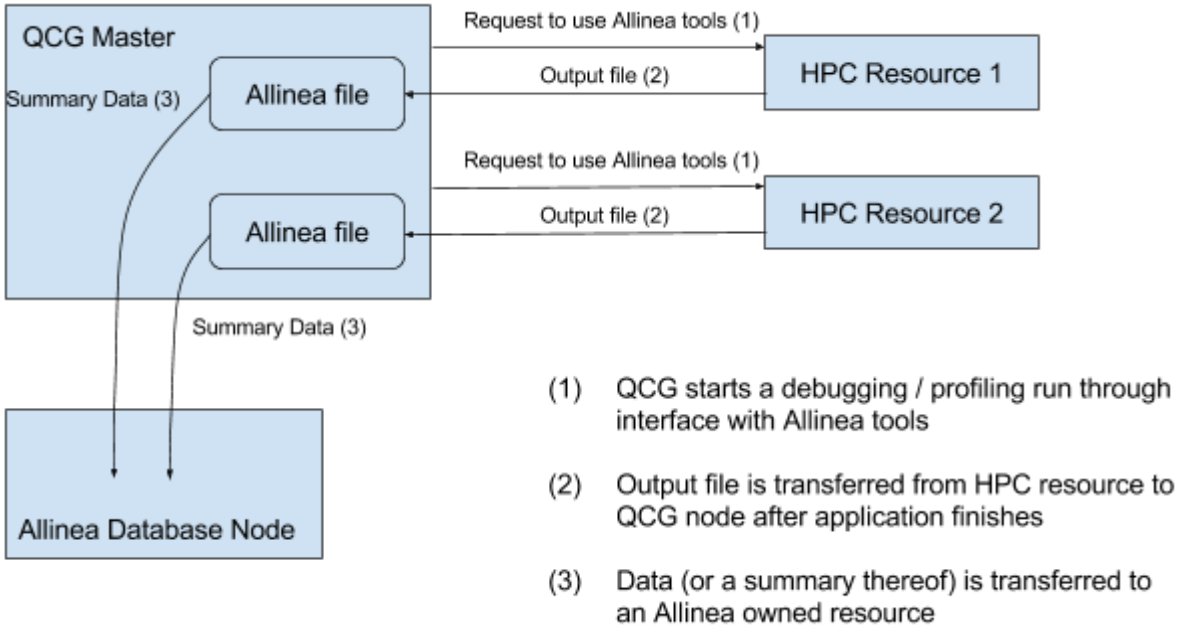


Figure 5: QCG master runs an application on an external HPC resource. Knowledge of how to run Allinea tools and collect output files is provided to QCG. A post processing phase on the QCG master node allows for Allinea output files to be transferred to an Allinea owned resource. A user may access Allinea data generated from this resource.

In addition to the documentation for launching jobs, Allinea will also need to specify the names of any output files that are generated that need to be transferred back to the QCG node, as well as defining a mechanism for forwarding on data to an Allinea owned resource. Step 3 outlined in Figure 5 is planned to be performed through the use of an Allinea script, which is to be run on the QCG host node upon completion of the user application and transfer of Allinea data from an external resource. In order that the script can be run at the correct time, it will be necessary for monitoring of the QCG-Client to check for its completion. This monitoring can be done via a script which will be developed in collaboration between Allinea and PSNC partners. A database is likely to be in the Allinea owned location where files are transferred to. This will be efficiently queried regarding the execution of the multiscale simulation, having stored summary data from Allinea output files (profiles and offline debug reports). Links to the Allinea files will be maintained in the database, allowing access via the database. Figure 5 depicts the flow of data when an offline debugging or profiling run is performed. In the case that an interactive debugging session is required, a slightly different approach is taken. An initial design is depicted in Figure 6.

In the scenario depicted in Figure 6 data is not required to be transferred back to the QCG host in an interactive debugging session. QCG will only start applications on the requested resources, and a user is responsible for connecting the applications to instances of the debugging GUI from their own machine. As the project progresses, closer integration with QCG is desired. In particular, the use of a feature of QCG called QCG Connect may enable a user to interactively debug applications without the need to know which system those applications have been submitted to. QCG Connect

allows for an interactive session to be established with a running task which was submitted via QCG. This is done by starting a shell session in the working directory of the task, from which any application can be started. All standard input and output streams are forwarded from the shell session to the QCG host, from where Allinea tools can capture this stream and use it for their own purposes. This is depicted in Figure 7.

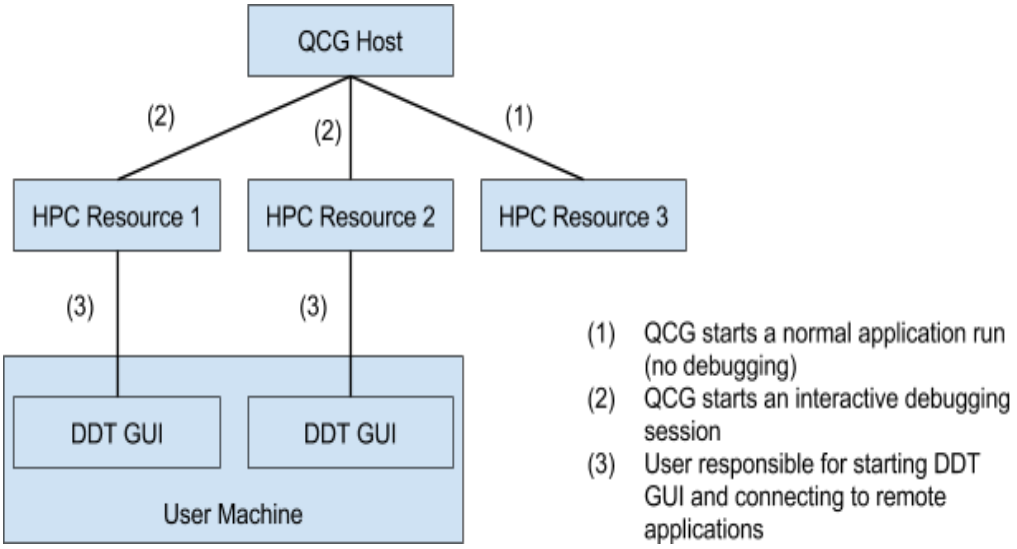


Figure 6: QCG launches applications on different resources. User connects DDT GUI to debug sessions running on each resource requested.

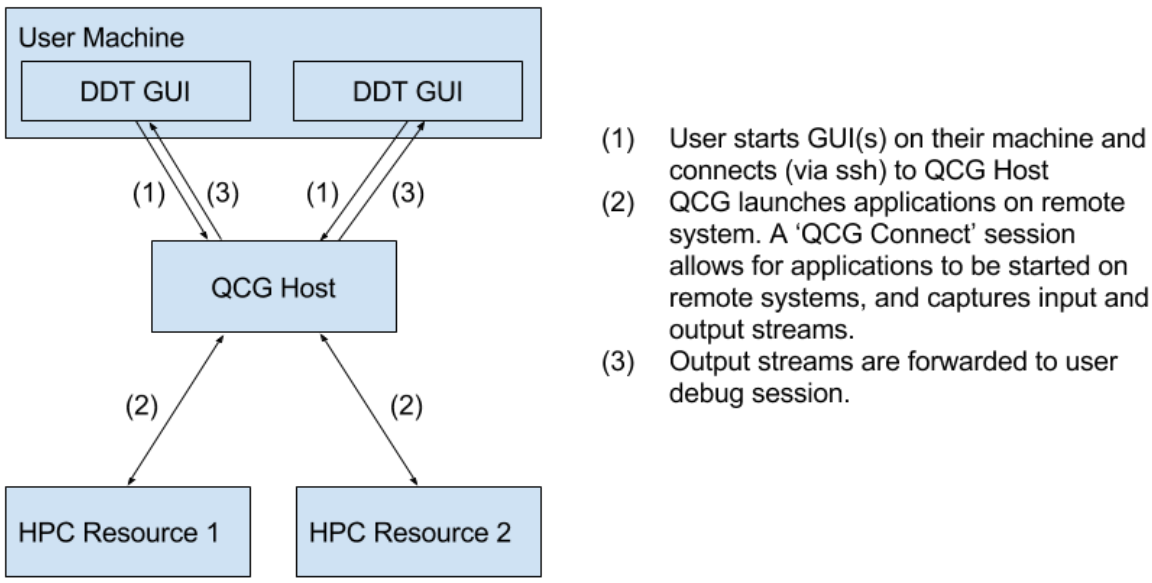


Figure 7: Future development will bring closer interaction between QCG and DDT. QCG Connect is used to create a connection to remote resources. A user then only has to connect a GUI to the QCG Host.

The features described in this section will simplify the process of debugging applications in a multiscale setting. The extensions to offline debugging allow for the fast identification of unexpected behaviour within a subset of applications run as part of a multiscale simulation. The integration of the

startup of interactive debugging sessions with QCG allows for a convenient method of controlling all or part of a multiscale simulation at run time.

3 Profiling with Allinea MAP

In this chapter the parallel profiling tool Allinea MAP (from now on MAP) is introduced. Section 3.1 describes how existing functionality is already applicable for developers of multiscale simulation codes. Section 3.2 outlines planned extensions to the profiling tool in order to better support the development and optimisation of multiscale simulations.

3.1 Current State

MAP is a parallel profiling tool which gives a breakdown of the recorded activity during a program run for programs written in C/C++ and FORTRAN. Figure 8 shows sample output from MAP, presented in the MAP GUI. Rich data is provided to the user and breaks a program down into CPU, I/O and MPI activity. Statistical sampling is used to inspect program state at irregular intervals. Samples are taken such that statistically significant events that occurred during a program run are observed, and a maximum of 1000 samples are reported to the user. This limit on the number of samples limits the amount of data that are generated, and allows for profiling to be highly scalable. MAP can be used currently in order to identify bottlenecks in single applications of a multiscale simulation. This allows for components of a simulation to be optimised, which is likely to lead to a more optimal multiscale simulation run. Work performed by partners at UvA as part of Work Package 2 has lead to a 30% efficiency gain in their single scale application modelling blood flow [5], and it is from these runs that the screenshot in Figure 8 was taken. As well as being able to optimise the performance of their code, MAP has enabled UvA partners to investigate the code in a third-party library which they are using, but are not the developers for. The understanding gained has allowed for third-party routines to be used in a more efficient manner.

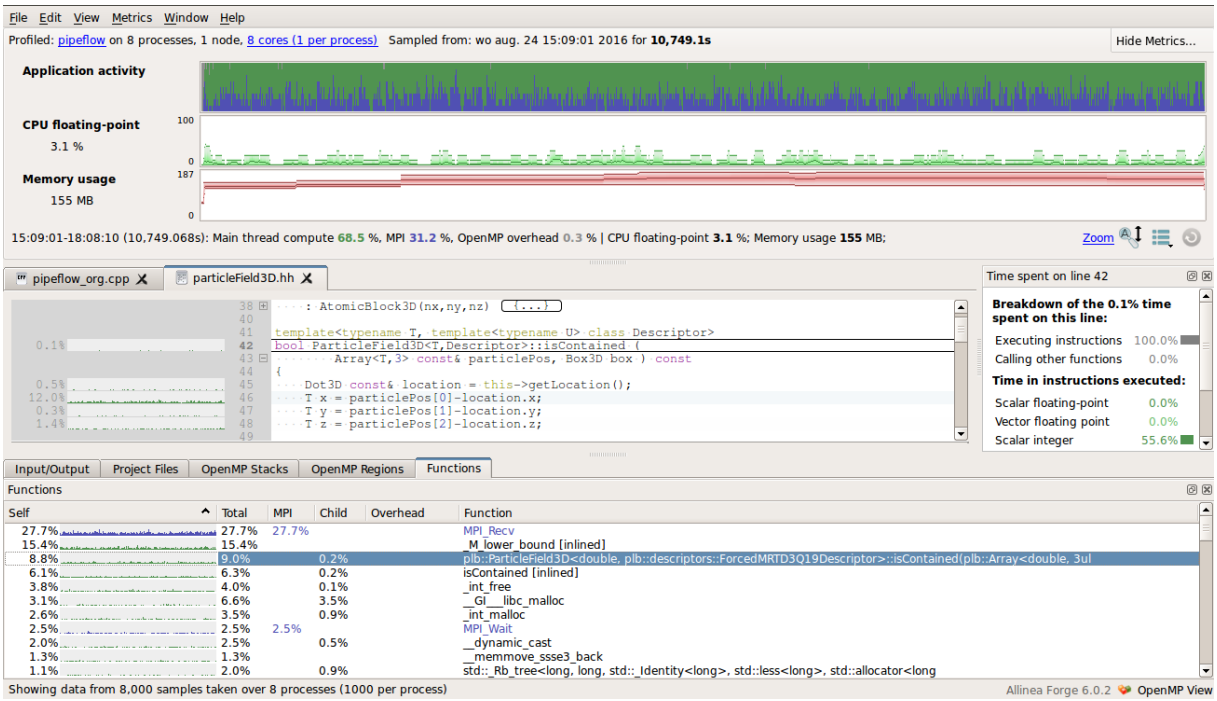


Figure 8: Example of a MAP profile. This was used by UvA partners to obtain a 30% increase in performance of a single scale application. The application activity spread across the processes is displayed, as well as attribution of measured time to individual functions and code lines.

Particularly salient for ComPat is the way in which MPI activity is reported in MAP. MPI is used for communication between processes in a single scale application. The MUSCLE2 framework [6] (see Section 2.4.1.1 of deliverable 5.1) is used for communication between processes in different applications. For MPI, MAP shows the time spent and the amount of data transferred in MPI function calls, which allows for intra-application communication patterns to be analysed and optimised. In order to optimise inter-application communication, similar data regarding the time spent and data volumes sent and received will also be reported for communication performed via MUSCLE2. From discussion with Work Package 3 partners, the following have been identified as salient metrics, although there is scope to add more if the need arises:

- Send rate (B/s) over MUSCLE
- Receive rate (B/s) over MUSCLE
- Time spent in MUSCLE function calls (s)
- The rate of MUSCLE function calls (calls/s)

As close collaboration with MUSCLE developers is maintained in this project, it is possible to change the MUSCLE library in order to maintain accounting information for MUSCLE2 function calls. This accounting information is to be read by MAP, but has the advantage of being available to users without the need of attaching a profiling tool. An alternative would be to intercept the MUSCLE calls through a profiling interface similar to that used in MPI, which again requires changes to the MUSCLE source code. Interception of the calls is also possible without changing the MUSCLE source code, although this approach would be more unstable than obtaining profiling data directly from the

library. Collaboration with MUSCLE developers has started, and it is expected that a prototype of reporting on the use of MUSCLE will be available by December 2016.

MAP will also be used in the project in order to monitor the energy usage of programs at run time. There are several possible sources for energy metrics from MAP, which are Intel's RAPL (Running Average Power Limit) on CPU counters and IPMI energy sensors. In order that the energy can be reported as part of ComPat it is required that all of the systems that are used can report the data from either RAPL or IPMI to MAP and also to Allinea Performance reports (see Section 4). Work has been performed in order to make sure that this information is available on the machines comprising the Experimental Execution Environment (see Deliverable 6.2) which uses compute clusters in Poznan, STFC and LRZ sites. On the Poznan system, an Allinea daemon which monitors system power has been installed. This retrieves IPMI energy information. On STFC's system work is ongoing in order to provide IPMI energy information to Allinea tools. This may not be done via a standard interface, but is going to be possible within the next few months. It may be necessary to report energy data out-of-band via an SNMP interface. Collaboration between Allinea and STFC is tracking the progress of this. Work at the LRZ system SuperMUC has resulted in the exposure of system monitored data being written in a format that is accessible to Allinea tools. This means that all existing energy reporting functionality will be available for programs run on all machines part of ComPat.

3.2 Extensions

In order to view all data related to a multiscale simulation, we propose a new interface for Allinea products. Summary information regarding a program run is to be stored in a database, which can be displayed to the user in graphical form through a web interface. The purpose of this interface is to present overview data of a multiscale simulation to a user without presenting an overwhelming amount. A subset of the data that is collected during the profiling of a multiscale simulation may be displayed. This can focus on the interaction between the applications rather than exposing the entire wealth of data collected to the user all at once. It is envisaged that the users will be able to configure which subsections of the data to display and how. One possible format for displaying of data to the user is shown in Figure 9, which takes the form of a graph. Other formats such as tables and lists may also be appropriate. As well as the overview data, detailed data regarding individual application runs will also be stored, with links to the locations stored in a database. Therefore, access to detailed performance data – such as a MAP profile or Performance Report (see Chapter 4) – can be gained on-demand.

Figure 9 shows an example graph for displaying a multiscale simulation to a user. This might use the run time of each application, together with the amount of computational resources consumed. This provides an overview of what application started and when, relative to others in the same multiscale simulation. The data to be displayed are stored to and retrieved from a database. Performance profiles with detailed data would be stored in a location accessible through a link stored

in the database. As part of ComPat, data and files to be stored would be exported to a database node in the manner depicted in Figure 5.

Alternatives for the displaying of multiple MAP profiles, such as merging these in the same interface similar to that shown in Figure 8 have been considered and rejected. Due to the richness of the MAP interface it would be difficult for a user to differentiate data from different applications, even for a modest number of applications. Another advantage of the approach proposed above is that it is not required for detailed information to be gathered for every application in a multiscale simulation in order to obtain high level overviews. These overviews already reveal interactions between different applications. With this in mind, Allinea plans to develop a lightweight sampler which records reduced information regarding a program's execution. This is to be a new product designed for minimal impact, whilst recording as much detail as possible. This will not provide insight into what actions an application is performing, but can be used to record the amount of data sent and received via MPI or a filesystem, as well as run time and resources consumed. Investigative work in this respect has already begun, and a prototype is anticipated before Milestone 16.2 (see Description of Actions) in Month 30, which is for a second release of the Profiling and Debugging tools.

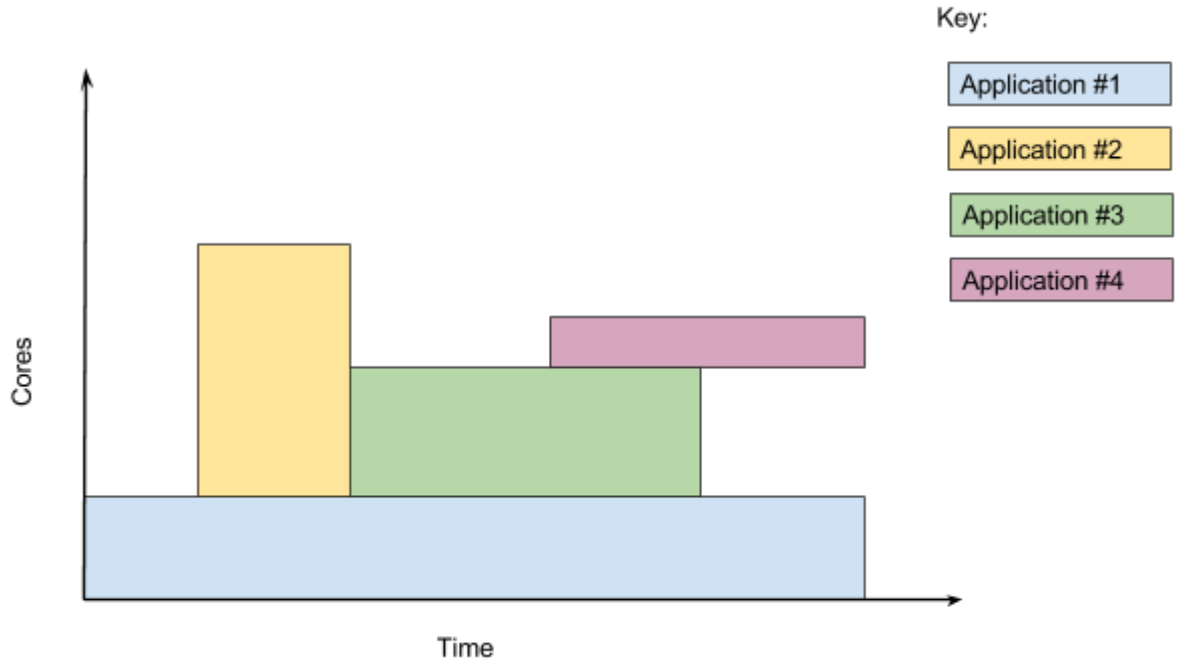


Figure 9: Example of a view on a web interface for displaying information from a multiscale simulation. The area of the blocks represents the core hours used by an application. Selecting an application allows access to more detailed information (including MAP profiles, Performance Reports and offline debug logs, if available).

Collecting information on the performance of MUSCLE calls within an application is an extension which has been discussed in the previous section. With this, it is possible for users to see for each individual application how efficiently they are using inter-application communication. The overhead (i.e. time spent waiting on communication to complete) depends on the performance of other

applications, so this information should be considered within the context of an entire multiscale simulation.

4 Performance Analysis with Allinea Performance Reports

In this chapter, the parallel profiling tool Allinea Performance Reports (from now on Performance Reports) is introduced. Section 4.1 describes how existing functionality is already applicable for developers of multiscale simulations. Section 4.2 outlines planned extensions to the profiling tool.

4.1 Current State

Performance Reports is a profiling tool which provides a summary of an MPI application run, as well as advice on how to optimise, targeting areas that cause the largest performance bottlenecks. Part of a Performance Report is shown in Figure 10. In addition to a summary of the time spent in parts of the program performing CPU, MPI or I/O activity, a Performance Report will advise on which areas to focus on, and what techniques to use in order to optimise performance. For example, in Figure 10 we see that due to the high proportion of memory accesses it is advised that the user improve cache performance. This information for a single scale application can be used as part of a multiscale simulation run in order to inspect the performance of component applications. Optimisation of the individual applications is likely to lead to better performance of the overall multiscale simulation.

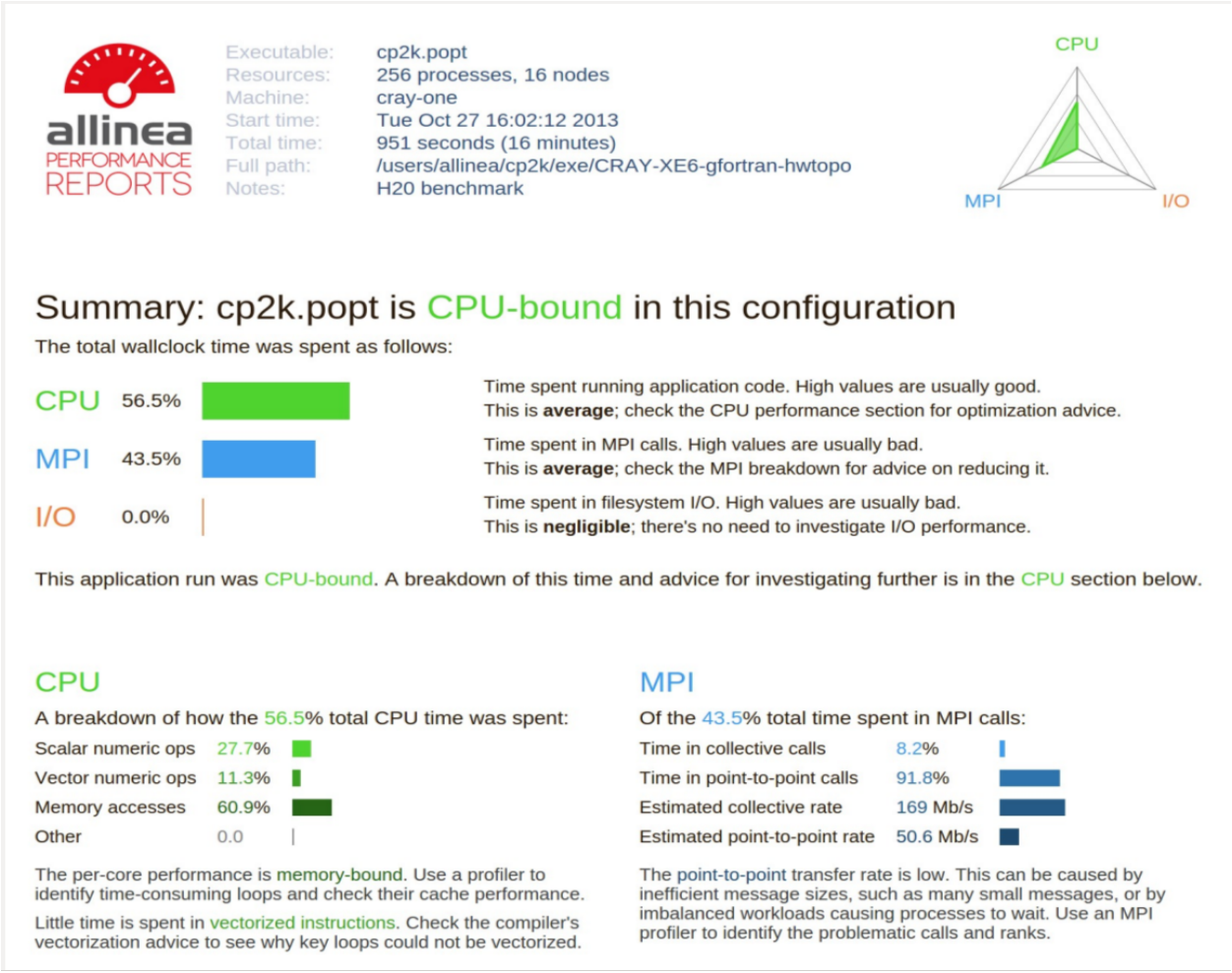


Figure 10: Selection of a Performance Report. A general overview of run time split into CPU, MPI and I/O parts is displayed. A breakdown of time spent in CPU, MPI, I/O, memory, energy and OpenMP performance is also given. Not all of this information is displayed in the figure to limit the length of the output.

4.2 Extensions

The extensions to Performance Reports are similar to the extensions to the profiler MAP. Support for reporting on inter-application communication via MUSCLE will be added to Performance Reports as it is added to the profiler MAP. The advice that is given regarding how to improve the performance of interactions across applications will develop as the understanding of efficient inter-application communication emerges during the course of the project. Close interaction with the application developers in Work Package 3 will be kept in order to develop this. Initial advice will be devised through interaction with the pattern developers in Work Package 2.

In order to view Performance Reports data as part of a multiscale simulation, access to a Performance Report will be provided through a web interface, an example output of which is provided in Figure 9. This is the same extension as described in the previous chapter for MAP. As part of the ComPat workflow, Performance Reports are to be transferred to a location in which it can be retrieved through the web interface via the mechanism depicted in Figure 5.

5 Conclusions

This report has given an overview of how Allinea's parallel debugging and profiling tools can be used in a multiscale environment. A high level discussion of the design of extensions to the tools to increase their efficacy for efficient software development in a multiscale environment has also been included. It was found that support needs to be added for Allinea tools to be used through another application interface (in this case the Work Package 5 middleware solution QCG). A new web-based user interface is also to be designed in order to provide a subset of large amounts of information collected during a multiscale simulation. Data is to be stored on a database node, with summary performance and debugging data stored in, and queried from, a database. Access to detailed performance and debugging information may be obtained upon request through this new interface. It is expected that a functioning prototype of the web interface will be available by March 2017. This is in time to meet Milestones 7 and 8.2, as outlined in Table 8 of the Description of Actions.

The implementation of the design outlined in this report is to be completed by Allinea, as the tools to be extended are owned by Allinea. If, during the course of ComPat, changes made to Allinea products outside of the scope of ComPat (such as through Allinea's support process) prove to be useful within the scope of the project, these improvements will be made available as they are developed. The design outlined here is an initial design which will be iteratively refined through a series of prototypes in collaboration with project partners, in particular application developers from Work Package 3. In terms of time scales for the completion of the extensions, initial plans have been made for features that will be available in the next 12 months (year two of the project). A prototype of the web interface (see Chapter 4) as well as the ability to conditionally generate offline debug reports (see Chapter 2), and integration with QCG tools are to be delivered to meet Milestone 8.2 in March 2017. These features will be further developed, and discussion with project partners will drive the way in which the usability and functionality is developed iteratively, which is to continue until month 24.

All of the work performed so far has been on-budget and on-time. Progress is as expected, and support will be provided for users as they begin to use the extensions to the tools when they become available.

6 Annexes

6.1 Allinea Commands

Command Line Prefix	Purpose	Output
ddt	Start an interactive debugging session with a GUI on the local system	No output file generated
ddt --offline	Start an offline debugging session	HTML debug report generated
ddt --connect	Start a debugger which will connect to an available parent process. This is used in order to connect to a GUI running on a remote system in a 'Reverse Connect' procedure. See Section 3.3 of the Allinea DDT User Guide **[1]** for more details.	No output file generated
map	Start a profiling session in a GUI on the current system	.map profile generated
map --profile	Start a profiling session without the need for a GUI	.map profile generated
perf-report	Start a profiling session without the need for a GUI	HTML summary profile generated

Table 1: List of command line prefixes for launching Allinea tools, along with a brief description of files that are generated.

7 References

[1] Allinea Software Ltd, "Allinea DDT User Guide," [Online]. Available: http://www.allinea.com/user-guide/forge/DDT.html#x8-27000II. [Accessed August 2016].

[2] B. Bosak, P. Kopta, K. Kurowski, T. Piontek and M. Mamoriski, "New QosCosGrid Middleware Capabilities and Its Integration with European e-Infrastructure," in *eScience on Distributed Computing Infrastructure*, Switzerland, Springer International Publishing, 2014, pp. 34-53.

[3] Poznan Supercomputing and Networking Center, "QCG Home Page," [Online]. Available: http://www.qoscosgrid.org/trac/qcg. [Accessed August 2016].

[4] D. Groen, A. Bhati, J. Suter, J. Hetherington, S. J. Zasada and P. V. Coveney, "FabSim: Facilitating computational research through automation on large-scale and distributed e-infrastructures," *Computer Physics Communications,* 2016.

[5] L. Mountrakis, E. Lorenz, O. Malaspinas, S. Alowayyed, B. Chopard and A. G. Hoekstra, "Parallel performance of an IB-LBM suspension simulation framework," *Journal of Computational Science,*

vol. 9, pp. 45-50, 2015.

[6] J. Borgdorff, M. Mamonski, B. Bosak, K. Kurowski, M. Ben Belgacem, B. Chopard, D. Groen, P. V. Coveney and A. G. Hoekstra, “Distributed Multiscale Computing with MUSCLE 2, the Multiscale Coupling Library and Environment,” *Journal of Computational Science,* vol. 5, pp. 719-731, 2014.