Deliverable 4.2: Report on status of performance profiling of multiscale simulations

Due Date	Month 18
Delivery	Month 18 + 1 week, PO agreed
Lead Partner	ARM
Dissemination Level	PU
Status	Final
Approved	Internal review yes
Version	V0.2

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
26.02.2017 v0.1	Oliver Perks	Outline of the report with headings of main sections
30.03.2017 V0.2	Keeran Babazon	ARM internal review

CONTRIBUTORS

Contributor	Role
Oliver Perks	WP4 Contributor, WP4 Leader
Keeran Brabazon	WP4 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 ARM (formerly 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 7: Ad-hoc capability for merging individual scale performance profiles to create one multiscale profile" and "Milestone 8.2: First version of the high level tools" have 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, has been marked as complete, with the delivery of Deliverable 4.1 [1].

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. This deliverable focuses on the level of support available for the different applications, and the modification of the tools and environment to provide this support. This work also extends to the integration with the multiscale pattern service of Work Package 2, the middleware services of Work Package 5 and the Experimental Execution Environment (EEE) of Work Package 6.

In Section 2 we provide an update on the Allinea tools, and the developments relevant to the ComPat environment, these updates mark the completion of Milestone 8.2. Section 3 introduces the application support for four of the ComPat applications, Fusion, SandBox, BAC and ChemShell. Section 4 presents an introduction to multiscale visualisation, and the completion of Milestone 7. Lastly, we outline the future work (Section 5) and conclude the deliverable (Section 6).

2. Tools Update for Multiscale Profiling

This sections presents an update on the use of Allinea tools [2] for profiling multiscale applications. Specifically, we focus on the performance profiling using existing Allinea tools, and not on debugging capabilities. As such we present an update on the two existing Allinea performance tools, MAP and Performance Reports. The latest version of these tools (v7.0.1) was released in advance of the month 18 review to constitute delivery of Milestone 8.2.

In Deliverable 4.1, we presented the current state of the tools, and the proposed enhancements to the tools to support the computing patterns and the environment unique to the ComPat project.

2.1. Allinea DDT

At this time in the project the majority of the effort to integrate Allinea tools into the ComPat stack has been to enable performance profiling, as opposed to debugging.

The Allinea DDT debugger can be launched in one of two modes. Firstly, an offline debugging mode – which does not allow for user interaction but generates a report on the detection of a crash. It can also provide some information on metrics such as memory leaks. This offline debugging can be integrated with the existing ComPat environment, in the same way that performance profilers are launched. Allinea DDT is used as a wrapper script around the `mpirun` command, which launches the application. In this scenario DDT is activated but does not display a GUI, or wait for user interaction, the application is just started asper normal, except with the debugger attached. As the launcher mechanism in DDT is the same as that in MAP the mechanisms shown in this document to attach MAP will also work for DDT offline debugging.

Whilst offline debugging can be used in some cases, for reporting crash information, the majority of debugging cases would require user interaction – which is not feasible in offline debugging. Thus only including support for offline debugging has some severe limitations. For increased capability we need to make use of interactive debugging, which presents a number of additional challenges. Currently a number of these capabilities have not been implemented to fully integrate into the ComPat environment. These have been previously discussed in Section 2.2 of Deliverable 4.1.

To perform interactive debugging, we need to be able to forward information from a debugging program to a GUI. When we are able to launch a GUI session we can enable debugging with relative ease. For example, we can integrate DDT launching into MUSCLE2 to debug a specific kernel of the simulation. This can already integrate with schedulers, such a Slurm [3], however the addition of QCG presents another ‘hop’ in the chain that we must forward data through. This capability is due to be developed in the latter part of the project, referred to as ‘deep track’.

2.2. Allinea MAP

A number of updates have been made to the Allinea MAP software stack, so support the ComPat environment. The majority of this work has focused on the support of the MUSCLE2 communication library. However, a number of other developments have beneficial applications to ComPat. Allinea MAP provides the primary profiling tool for applications, however, developments to the Allinea Performance Reports product are documented in Section 2.3.

2.2.1. MUSCLE2 Custom Metric

As part of the support for the ComPat environment it became crucial to provide some support for the MUSCLE2 communication library. This library allows multiple MPI applications to communicate with each other. This forms the basis on a number of the applications within the computational patterns profile.

In Deliverable 4.1 we identified a set of key performance metrics for capturing the performance of an application with MUSCLE2 integration. We reproduce that initial list below:

- 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)

To generate this custom metric, we had to modify the MUSCLE2 library, to provide the required software performance counters, and expose them to the Allinea MAP custom metrics interface. These changes have been integrated into the MUSCLE2 github repository [4], within the ‘compat’ branch – and are enabled at compile time through the performance ‘-p’ flag.

The custom metric is then built against this new MUSCLE2 library, and installed locally in a user’s home space. The set of metrics made available through this library are listed below:

- MUSCLE2 sent rate (B/s)
- MUSCLE2 send calls (/s)
- MUSCLE2 send duration (s)
- MUSCLE2 receive rate (B/s)
- MUSCLE2 receive calls (/s)
- MUSCLE2 receive duration (s)
- MUSCLE2 barrier calls (/s)
- MUSCLE2 barrier duration (s)

This data is presented in a very similar manner to that of MPI communications, in a process centric way. However, this can cause some issue with MPI applications, as a current limitation of MUSCLE2 is that only the master MPI process (rank 0) can communicate. For a large scale parallel job this means that activity within MUSCLE2 is generally only reported on a single thread, the activity of the other threads is attributed to other behaviour. This will most often be an MPI call, due to the need for synchronisation.

Visualising MUSCLE2 performance data in Allinea MAP (see Figure 1) therefore shows a single rank 'max value' (highlighted by the top red point of the MUSCLE2 receive duration graph) and all other ranks reporting 0s. This naturally impacts the derived metrics, such as mean and standard deviation. However, this is actually representative of the underlying behaviour of the application.

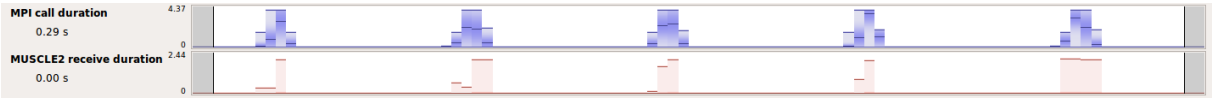

Figure 1: MPI vs MUSCLE2 call duration

From Figure 1 we can see the intertwined behaviour of MUSCLE2 and MPI. Whilst only a single MPI rank is communicating over MUSCLE2, all other processes must wait at the next synchronisation point, most likely MPI. This creates a natural duality between the two libraries, represented by their call durations.

The exception to this behaviour is the MUSCLE2 barrier call, as this is executed by all processes, within the MPI job. This means that the activity time is split between the different calls, the associate MUSCLE2 communication for the MUSCLE rank and the MUSCLE2 barrier for the other ranks.

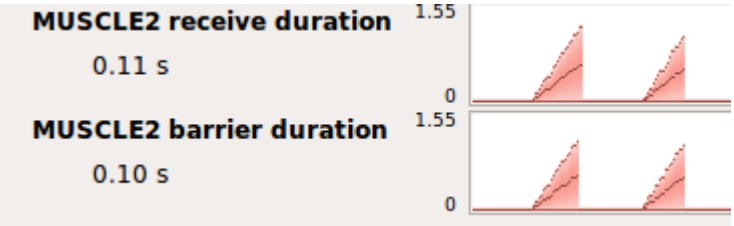

Figure 2: MUSCLE2 activity distribution

Figure 2 shows the case when we perform a MUSCLE2 barrier, before conducting the MPI communication. Here we have two processes, one performing the MUSCLE2 receive and the other waiting, in a MUSCLE2 barrier, hence the call duration is equal and the time split is between the different ranks.

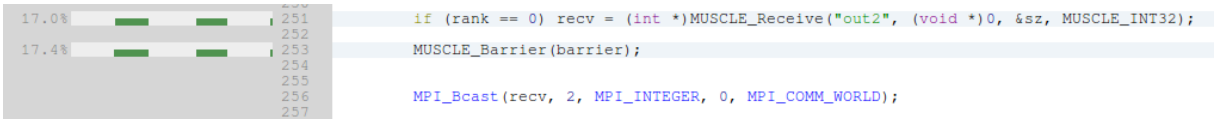

Figure 3: MUSCLE2 source code for behaviour in Figure 2

The code for this example is shown in Figure 3, from which Figure 2 was generated. Here we see that rank 0 performs the MUSCLE2 receive operation. The other MPI ranks can progress, normally to an MPI call – however, in this case we make use of the MUSCLE2 barrier to pause all ranks, before progressing to the MPI call, together. The motivation behind this behaviour is in the nature of the MUSCLE2 barrier, as it makes use of a passive wait, as opposed to the active wait used within MPI. This means when the MPI broadcast (line 256 in Figure 3) is executed, it acts as a synchronisation point, where all ranks wait for rank 0 to finish the MUSCLE2 call. This active wait is constantly cycling the CPU to enable it to respond as quickly as possible. In the case of MUSCLE2 we are less concerned with this reaction speed, as the communications may be very slow, as the end points of communication may be compute resources in different countries. Hence it performance a passive wait, polling periodically for an update on rank status. This can save significant amounts of energy, if all processes of a large parallel job are waiting for a long time. However, they might be slightly slower to react when the communicating thread does return.

2.2.2. JSON Export

One of the major additions to the Allinea MAP tools has been support for JSON export of MAP performance data. This feature is designed to export the specific data contained within the MAP file in an open machine readable format.

Specifically, a MAP profile will contain a number of samples. For each sample, the tool stores a set of values for each metric collected. Because we do not store the metric value for each process in a job, we instead store a fixed number of aggregated metric values. For example, for a metric, we aggregate across the metric value for every process and we store the maximum sample value, the minimum, the mean, the standard deviation and the sum.

The JSON export feature allows the user to obtain this raw data, for each of the metrics collected. So for each metric collected the JSON file will contain up to 5000 values, 1000 sample value for each of the 5 different aggregation metrics discussed above.

2.3. Allinea Performance Reports

Performance Reports forms a secondary profiling tool for the ComPat project. It differs from Allinea MAP by providing single aggregated values for metrics of interest, as opposed to a temporal based analyses. This is useful for storing performance data (as a web page, as part of the output files), and for integrating back into the performance history, for prediction purposes, of application kernels.

2.3.1. Partial Reports

In addition to the MUSCLE2 custom metric for MAP Allinea Performance Reports has been extended to support data from custom metrics. With the case of the MUSCLE2 we have developed a partial report to display the aggregated metrics in Performance Reports. Performance Reports is designed to present a single aggregated value for a metric.

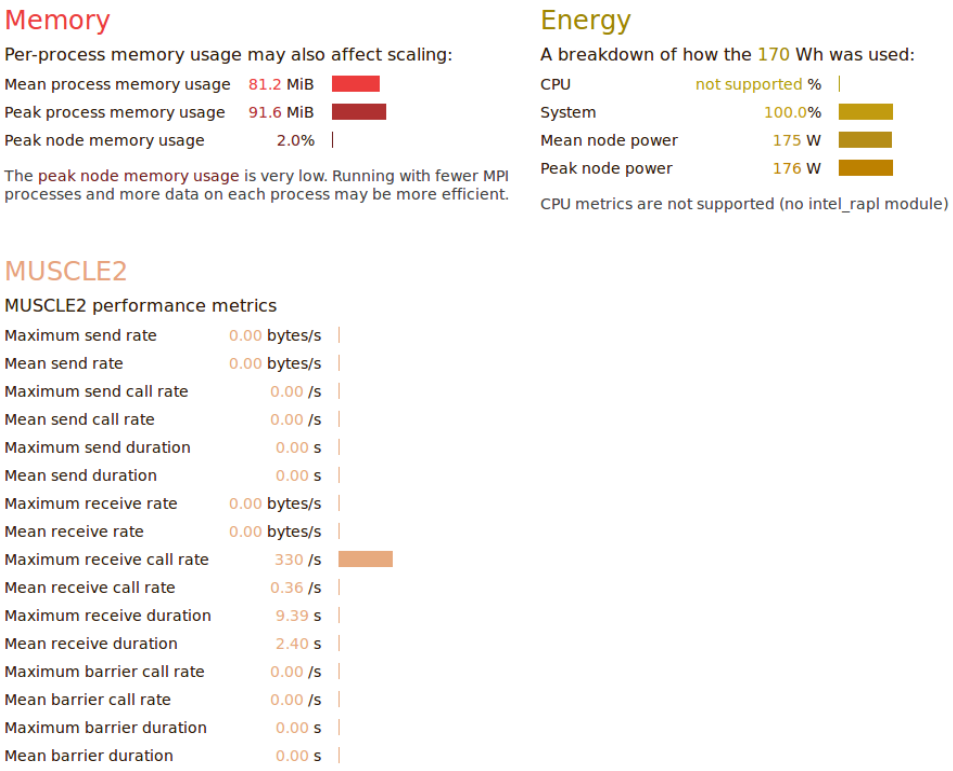

Figure 4: MUSCLE2 partial report in Allinea Performance Reports

In Figure 4 we show an example MUSCLE2 partial report displayed in Performance Reports, alongside other performance metrics. This data was collected from the large kernel as part of the tube example from the sandbox multiscale application, we cover this application in more detail in Section 3.2.

With the case of MUSCLE2 metrics we have taken an approach to aggregation which better represents the usage model of MUSCLE2. When you have a MUSCLE2 send or receive call it is only performed from a single MPI rank (rank 0), regardless of how many MPI ranks are active within the job. With the custom metric we record the send and receive activity – on all MPI ranks, despite only one being active. Thus when it comes to aggregating this data we have two phases of aggregation – across the ranks and then across the samples.

With MUSCLE2 send and receive calls we need to take the maximum values across the ranks and then calculate the mean (or max) across the samples. This means that as the number of MPI ranks scales the mean will not be impacted by this – and will only report on the activity of rank 0.
However, in the case of MUSCLE2 barrier this can be called by all MPI ranks, thus scaling the MPI ranks will have an impact on the barrier metrics, so we must take a mean aggregation across the ranks and a mean or max across the samples.

This capability forms the first step of the expert advice framework for ComPat applications – as described by Task 4.3.

2.3.2. JSON Export

Similar to the JSON export capability of Allinea MAP we have developed a similar capability for Allinea Performance Reports. This facilitates the export of the aggregated metric data to a machine readable format, in a similar layout to that of MAP. The export capability of the two tools is complimentary to the extent that the data can be consumed by third party tools to provide data analytics from multi source data.

The export capability for Performance Reports is of particular relevance to the ComPat project, as the aggregated data can feed directly into the ComPat tool set, specifically performance matrix component of the pattern service. The performance data collected for multiscale simulations can be recaptured, and used to help performance prediction and optimisation criterion when submitting a QCG job, this is covered in more detail in Section 2.4.3.

2.4. Integration with ComPat Tools

2.4.1. MUSCLE2

MUSCLE2 is a key component in many of the multiscale model applications, providing a means for multiple kernels to be executed in parallel and communicate. As such it is important for the Allinea tools to integrate with MUSCLE2.

```
small = NativeInstance.new('small', 'map', args:'--profile -o=small.map mpirun -np 2 '"#{dir}/tube_muscle2_small")
small2 = NativeInstance.new('small2', 'map', args:'--profile -o=small2.map mpirun -np 2 '"#{dir}/tube_muscle2_small2")
large = NativeInstance.new('large', 'map', args:'--profile -o=large.map mpirun -np 4 '"#{dir}/tube_muscle2_large")

#small = MPIInstance.new('small', "#{dir}/tube_muscle2_small",{:mpiexec_args => '-np 2'})
#small2 = MPIInstance.new('small2', "#{dir}/tube_muscle2_small2",{:mpiexec_args => '-np 2'})
#large = MPIInstance.new('large', "#{dir}/tube_muscle2_large",{:mpiexec_args => '-np 4'})
```

Figure 5: Allinea MAP integration with MUSCLE2 native instance

In Figure 5 we show how a native instance can be used to launch Allinea. Specifically, we are launching three instances of MAP, one for each kernel – thus we are profiling all of the kernels simultaneously as if they were standalone applications. Here the MUSCLE2 framework is calling MAP from within the parallel job – telling it to do offline profiling and generate a specific MAP file named by the kernel name.

2.4.2. QCG (and the EEE)

QCG is used within ComPat as the brokering service to select the resources to submit to. It acts as a meta-scheduler to provide a consistency over the different schedulers used on the different HPC platforms.

Integrating Allinea tools with QCG can be done in different ways. Firstly, we focus on the so called off-line profiling. That is to launch the profiler (either Allinea MAP or Performance Reports) and collect results without any further user interaction.
An online capability would be required to perform interactive debugging of an application. This capability requires tighter integration and has not been fully explored.

```
#QCG name=ollyTest
#QCG host=eagle.man.poznan.pl
#QCG nodes=1:14
#QCG grant=plgoperks2016a
#QCG output=${JOB_ID}.out
#QCG error=${JOB_ID}.err
#QCG queue=plgrid
# Change directory to the working directory
export WORK_DIR=/home/plgrid/plgoperks/FluidFlow/workflow

module use /home/plgrid-groups/plggcompat/Modules/
module load compat/common/allinea/7.0 compat/common/muscle2/eagle/compat-1.2 openmpi/1.8.1-1_gcc482

cd ${WORK_DIR}

muscle2 -c tubemap.cxa.rb -m large small small2

cp *.map ${QCG_JOB_DIR}

#QCG stage-out-file=large.map -> ${JOB_ID}_large.map
#QCG stage-out-file=small.map -> ${JOB_ID}_small.map
#QCG stage-out-file=small2.map -> ${JOB_ID}_small2.map
```

Figure 6: QCG script to launch MUSCLE2 running with Allinea MAP – shown in Figure 5

Figure 6 shows how QCG can be used to launch a MUSCLE2 application on the EEE, in this example we make use of the SandBox application. This specific example is using MUSCLE2 to profile the three different application kernels with Allinea MAP, as shown in Figure 5.

At the end of the profiled kernel, MAP will generate the output file (e.g. small.map), the QCG script moves these files to the QCG working directory and instructs the job to stage these files back out to the submission machine.

2.4.3. Pattern Service

The pattern service is responsible for generating the description of the multiscale model job to pass to QCG. Part of this process is providing the performance information for the components of the job, to allow QCG to broker for resources as optimally as possible.

For this purpose, a performance matrix is used. This details the performance characteristics of the kernels under different environment conditions (node type, core counts, problem size). The role of Allinea tools is to provide some of the data to populate this performance matrix. Currently, the primary focus of brokering is to reduce the runtime of an application, with an intention to start including information for energy usage. For this it is simple to use Allinea Performance Reports to collect information from application runs, and feed in back into the pattern service.

A simple feedback loop could be used, based on the JSON export of Performance Reports, to feed the performance information of a kernel, which has just finished running, back into the performance matrix. This level of integration would require some additional engineering, to correctly capture the kernel

information (from the pattern service or MUSCLE2) and the system information (from QCG), and store the data in a central data store.

3. Application Profiling with Allinea Tools

In this section we detail the progress that has been made to profile the ComPat multiscale applications with the Allinea toolset. The nuances of each application present different challenges to profiling, as such, we present the modifications made to enable their profiling.

3.1. Fusion Application

The fusion application is a MUSCLE2 based extreme scale demonstrator. The Allinea tool support has been provided via the MUSCLE2 integration (Section 2.4.1) and the MUSCLE2 custom metric (Section 2.2.1). The application simulates the time evolution of electron and ion temperature profiles (macro) in the core of the plasma and the effects of turbulence at micro scales. It consists of 3 submodels (transport, equilibrium and turbulence) and one numerical tool (to convert heat fluxes coming from the turbulence submodel into coefficients taken into account by the transport submodel to evolve the temperature profiles). The turbulence code (GEM, 3D gyrofluid fluxtube approximation) is the primary submodel and all other codes (ETS for 1D transport, CHEASE for 2D equilibrium and imp4dv to convert fluxes) are auxiliaries in this extreme scaling scenario. The initial plasma state corresponds to a discharge on the ASDEX Upgrade Tokamak in IPP Garching.

In this example all four submodels are profiled by Allinea MAP simultaneously. The GEM kernel is MPI based which is scaled with core count, whereas the three auxiliary models are executed in serial in support of the turbulence code. The profile shown in Figure 8 shows the primary kernel of the fusion application. What we see from this profile is that the vast majority of the application time is spent computing and communicating with MPI – and not with MUSCLE2. As MUSCLE2 calls only take place from a single MPI rank it would be very detrimental to system efficiency if all MPI ranks are waiting for the MUSCLE2 rank to receive data. In this example this synchronisation overhead is represented by the time spent in MPI calls – which aligns with the MUSCLE2 receive duration.

In Figure 7 we present the profile for one of the three supporting kernels of the fusion application, specifically the Chease kernel. In this scenario the kernel is run in serial – and spends most of its time waiting for data – as shown in the saw tooth profile of the call duration time. We can see that minimal time is actually spent computing, which is acceptable in this case as it is a serial application running on a small amount of computing resources. Chease feeds the primary model with data. As such, a loss of efficiency for the sub model is acceptable to reduce the overall cost of the parallel primary model.

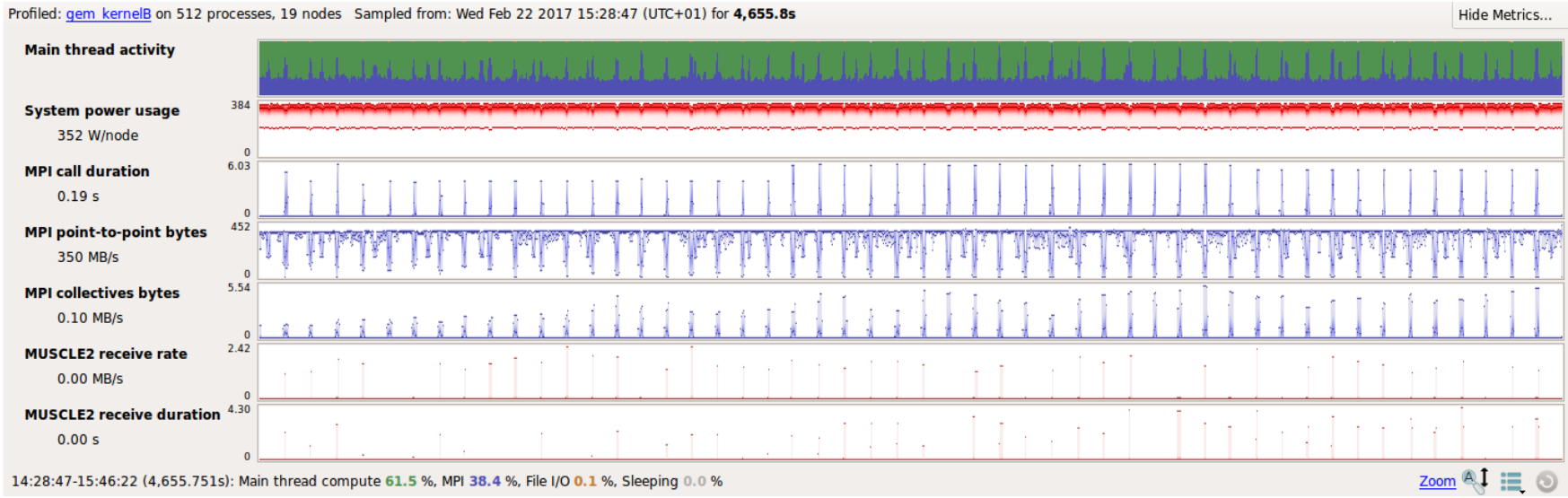

Figure 8: The Gem kernel in the Fusion application on 512 cores

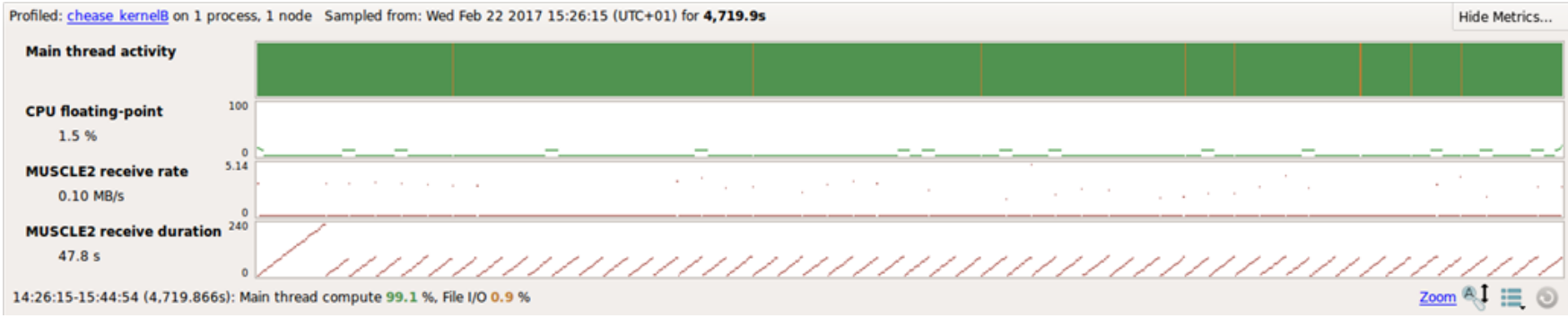

Figure 7: MUSCLE2 profile of Gem kernel in support of simulation in Figure 8

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

3.2. SandBox

The SandBox application is another MUSCLE2 based application. It makes use of the Palabos Lattice Boltzmann library. For testing purposes, we have been making use of the tube example which runs three kernels in parallel – a large and two small kernels. In this multiscale model we consider a cell-based blood suspension model as the primary model, coupled to continuous LBM blood flow models as the auxiliary models. This multiscale model mimics the real situation of virtual artery - under development in at the University of Amsterdam. This model is used as a proof of concept of the ideas of pattern service by changing some simulation parameters. The current test case uses 128^3 and 32^3 lattice size for the primary and auxiliary models.

Allinea tools support for SandBox is achieved in a very similar way to the fusion application (Section 3.1). In Figure 10, Figure 11 and Figure 12 we show the profiling capability of the Allinea tools on the sandbox application. Specifically, we focus on the MUSCLE2 call time, and the MPI call time, as this highlights the interplay between the ranks within a kernel and the different kernels. We note how the large kernel starts with a processing phase while the other two kernels await data. The two small kernels then perform their operation before the whole application closes. In this example only a single motif of the model is executed, though it is expected that multiple motifs are executed, thus a more cyclic behaviour is expected, such as the behaviour seen in the fusion application in Section 3.1.

One interesting artefact shown in the large kernel is the computational resource for the MUSCLE2 receive call – as we can see it is spending all of its time in memory operations – as it is polling a variable to check if it is ready to progress.

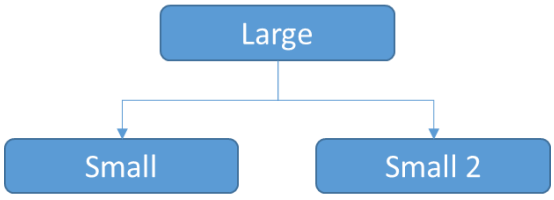

Figure 9: SandBox Tube example kernel connectivity

Note that in this example not all of the kernels run for the entire duration, and so the timelines cannot be compared directly, as they are presented as individual application profile. The combination of these profiles, into a cohesive multiscale simulation visualisation is a separate topic of research which we present in Section 4. The layout of the kernels, in terms of processing dependency, is depicted in Figure 9.

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

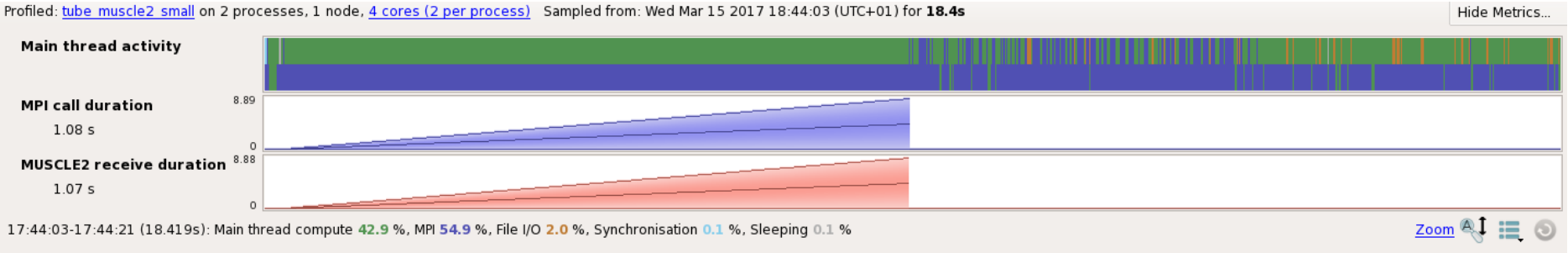

Figure 10: SandBox Tube example profile of the small kernel

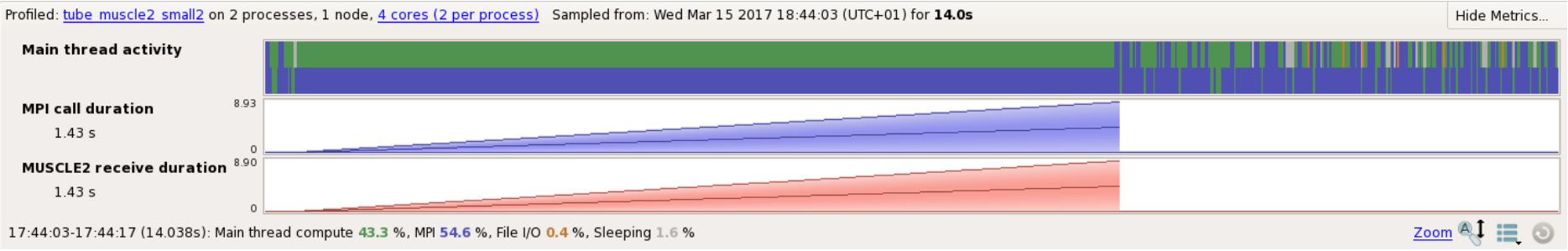

Figure 11: SandBox Tube example profile of the small2 kernel

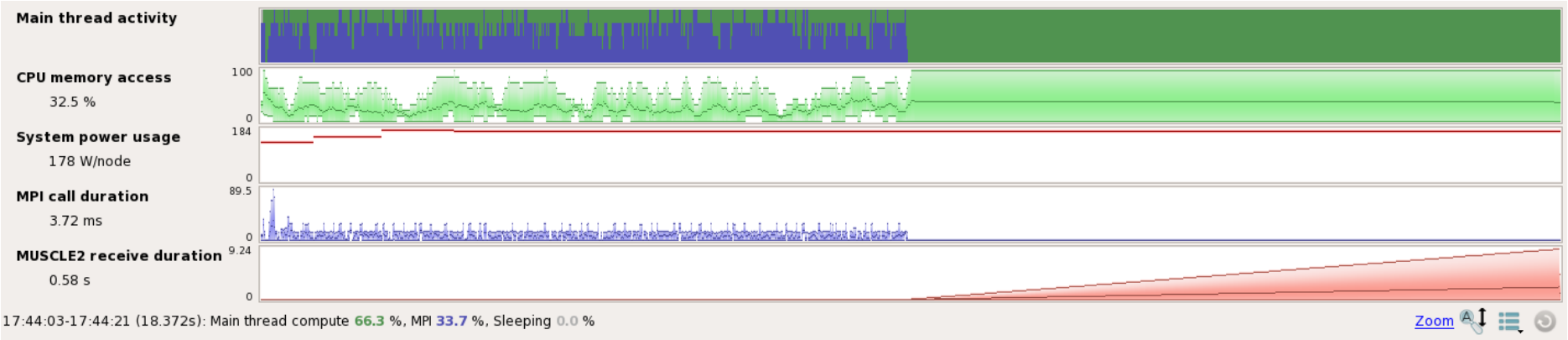

Figure 12: SandBox Tube example profile of the large kernel

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

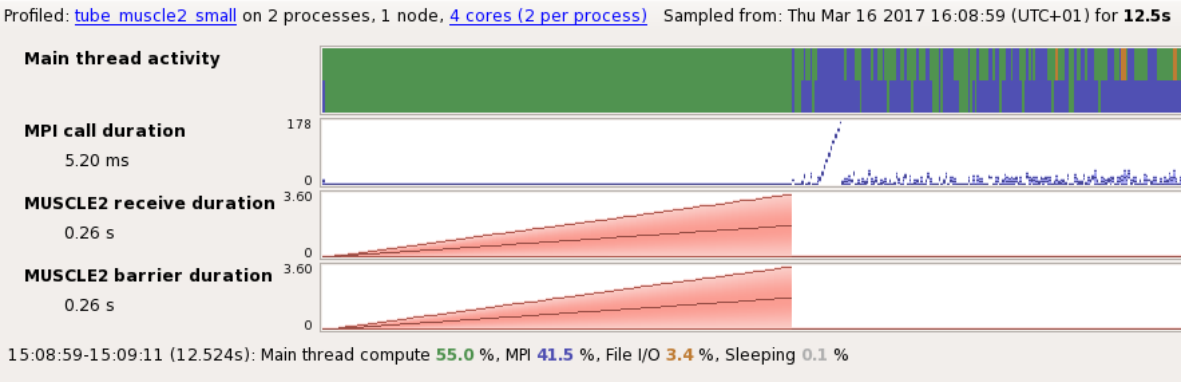

Figure 13: Modified SandBox small kernel to use a MUSCLE2 barrier before MPI communications

In Figure 13 we show the profile of a modified version of the small kernel, as opposed to the original presented in Figure 10. In this example we have inserted a MUSCLE2 barrier just after the MUSCLE2 receive. This forces the MPI process to wait in MUSCLE2 before it can progress. This means the time is reported as MUSCLE2 cost rather than an MPI cost, as we are changing where the synchronisation is attributed to. As discussed in Section 2.2.1 the advantage of using a MUSCLE2 barrier rather than an MPI synchronisation is that MUSCLE2 can make use of an idle wait, as opposed to a busy wait, and potentially reduce the energy consumption of the idle portion of a kernel.

3.3. Binding Affinity Calculator (BAC)

The Binding Affinity Calculator (BAC) is a molecular dynamic code based on the NAMD [5] scientific library and AMBER [6]. NAMD itself is based in the CHARM++ parallelisation library, as opposed to MPI. Allinea has traditionally provided support for MPI based parallel applications, and does not have direct support for native CHARM++ [7].

To support profiling of the BAC application in Allinea tools, we have had to reconfigure CHARM++ to make use of MPI as the underlying communication layer. However, this requires that the NAMD library must be rebuilt against this new version of CHARM++, and the use of MPI as the underlying communication layer may have some impact on availability of features.

BAC makes use of the replica computating model. As such it launches multiple applications in series, which are profiled using Allinea MAP. Unlike the MUSCLE2 based applications, BAC passes information between models using files. Thus, file I/O can be monitored as a proxy for inter-model communication cost.

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

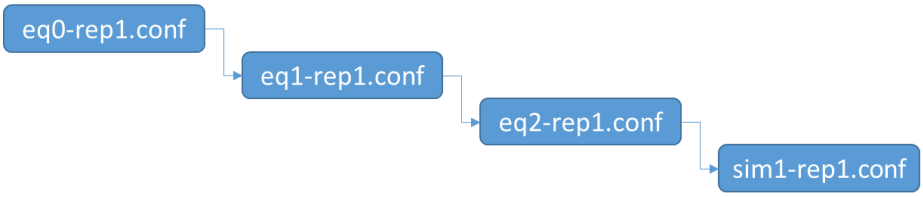

Figure 14: BAC 4 replica example connectivity flow

In Figure 14 we present the connectivity between the multiscale replicas, and the flow of information between them, in our four replica example. Here we have four replicas run consecutively, with no information transfer during each application instance, only from the end of one replica to the start of the next.

```
NAMD_DIR=/tmp/lustre_shared/plgkbrabazon/NAMD/NAMD_2.12_Source/Linux-x86_64-g++
CHARM_CMD=${NAMD_DIR}/charmrun
EXEC_NUM=0

mkdir -p map_profiles
PROFILE_DIR=$(pwd)/map_profiles
export ALLINEA_CHARM_TOOL_CMD=map
export ALLINEA_CHARM_TOOL_ARGS="--profile -o ${PROFILE_DIR}/app${EXEC_NUM}.map"

NAMD_CMD="${CHARM_CMD} +p${SLURM_NTASKS} ${NAMD_DIR}/namd2"

echo "current step $REP ..."

cd input/mineq_confs
${NAMD_CMD} eq0-rep${REP}.conf > ../replicas/rep${REP}/equilibration/eq0.log
let "EXEC_NUM++"
export ALLINEA_CHARM_TOOL_ARGS="--profile -o ${PROFILE_DIR}/app${EXEC_NUM}.map"

${NAMD_CMD} eq1-rep${REP}.conf > ../replicas/rep${REP}/equilibration/eq1.log
let "EXEC_NUM++"
export ALLINEA_CHARM_TOOL_ARGS="--profile -o ${PROFILE_DIR}/app${EXEC_NUM}.map"
```

Figure 15: QCG run script fragment for running NAMD with Allinea MAP

```
mpirun_cmd=`which mpirun 2>/dev/null`
if test -n "$mpirun_cmd"
then
  [ -n "$MPI_MACHINEFILE" ] && args=" -machinefile $MPI_MACHINEFILE $args"
  setarch_cmd=`which setarch 2>/dev/null`
  if [ -n "$setarch_cmd" -a -x "$setarch_cmd" ]
  then
    # Disables randomization of the virtual address  space  (turns  on
    #          ADDR_NO_RANDOMIZE).
    cur_arch=`uname -m`
    test $QUIET -eq 0 && echo "charmrun>  $setarch_cmd $cur_arch -R  $ALLINEA_CHARM_TOOL_CMD $ALLINEA_CHARM_TOOL_ARGS mpirun -np $pes $args"
    $setarch_cmd $cur_arch -R  $ALLINEA_CHARM_TOOL_CMD $ALLINEA_CHARM_TOOL_ARGS mpirun -np $pes $args
  else
    test $QUIET -eq 0 && echo "charmrun> $ALLINEA_CHARM_TOOL_CMD $ALLINEA_CHARM_TOOL_ARGS mpirun -np $pes $args"
    $ALLINEA_CHARM_TOOL_CMD $ALLINEA_CHARM_TOOL_ARGS mpirun -np $pes $args
  fi
else
  mpiexec_cmd=`which mpiexec 2>/dev/null`
  if test -n "$mpiexec_cmd"
  then
    test $QUIET -eq 0 && echo "charmrun> $mpiexec_cmd -n $pes $args"
    test $QUIET -eq 0 && echo
    "$mpiexec_cmd" -n $pes $args
  else
    echo "Don't know how to run MPI program."
    exit 1
  fi
fi
```

Figure 16: Charm++ run script to profile with Allinea MAP

To enable the profiling of the BAC application we must configure Charm++ to insert environment variables into the launch script before the ‘mpirun’ command. These environment variables are

generated and controlled by QCG. Shown in Figure 15, we present a fragment from the QCG submission script, this sets the `ALLINEA_CHARM_TOOL_CMD` and `ALLINEA_CHARM_TOOL_ARGS` environment variables which are picked up by our customised Charm++ run script, shown in Figure 16.

In Figure 17 we present an Allinea MAP profile of one of the BAC kernels, running NAMD. In this case we are executing a single replica, formed of four NAMD kernels and one Amber kernel. In this case we are displaying the `*eq2-rep1.conf* ` NAMD configuration, which is the third of our simulation of four NAMD kernels. As discussed previously we see two small I/O phases, a read phase at the start and a write phase at the end – to obtain and distribute data, respectively. Beyond these phases nearly no I/O takes place, thus clearly distinguishing them.

The BAC application presents an additional point of interest for profiling with Allinea tools, as it operates a task based programming paradigm. A traditional application profile assumes that all ranks are operating in an approximation of lock-step progression. This is not the case with BAC, as each rank is operating independently. This is represented in Figure 18, where we see the distribution of work is roughly split across four work queues performing two functions.

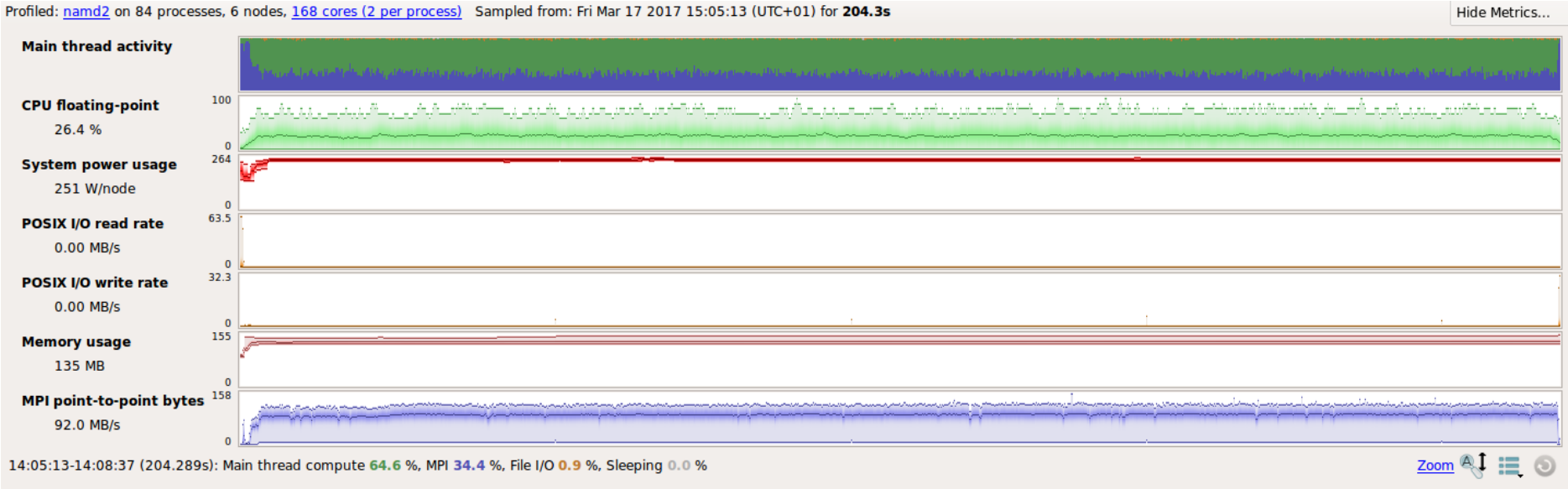

Figure 17: BAC 3rd of 4 NAMD kernels in the replica

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

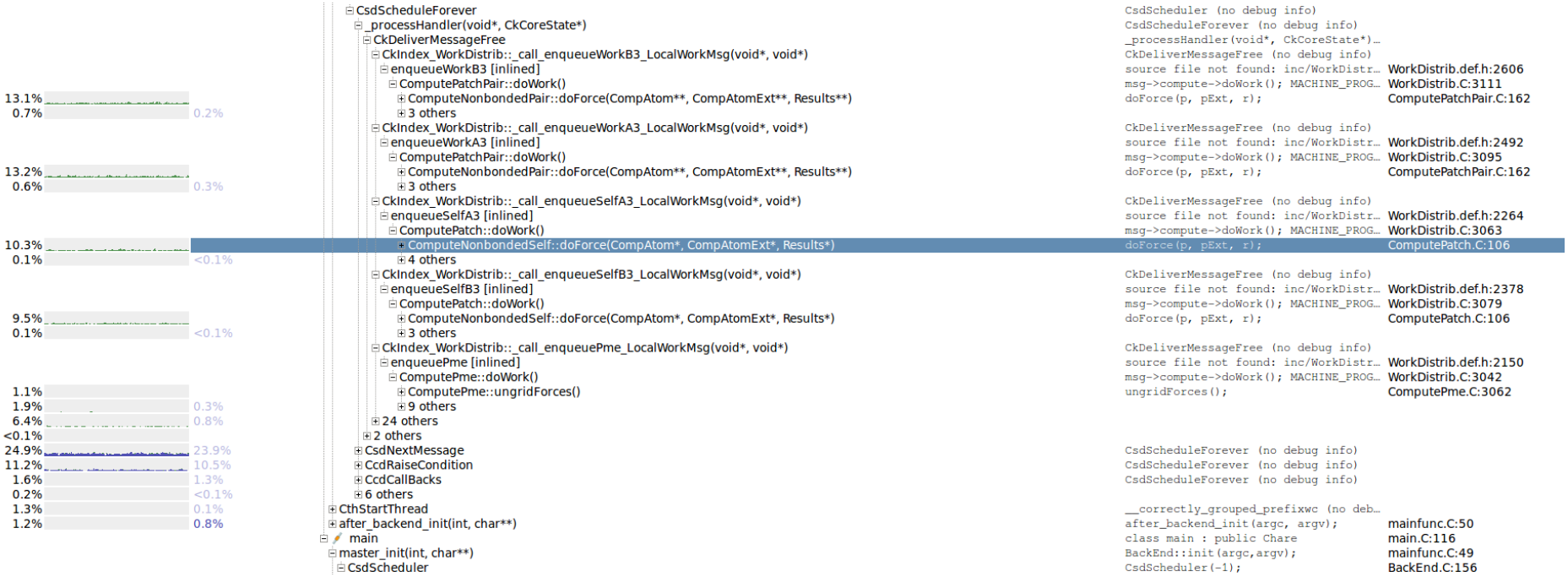

Figure 18: Work distribution of BAC tasks

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

3.4. ChemShell

ChemShell is a scriptable computational chemistry environment with a particular strength for combined quantum mechanical and molecular mechanical (QM/MM) simulations and is widely used for both materials modelling and biochemical calculations. The test cases include a typical bioactive molecule C10H11N5O5PS and a typical embedded MgO cluster (251 atoms) employing QM code GAMESS-UK [8] and MM code GULP [9].

The ChemShell application is an interesting application to profile within the Allinea tools. It is a framework which connects kernels of different chemistry packages together to solve a wider problem, as such, it is itself a framework.

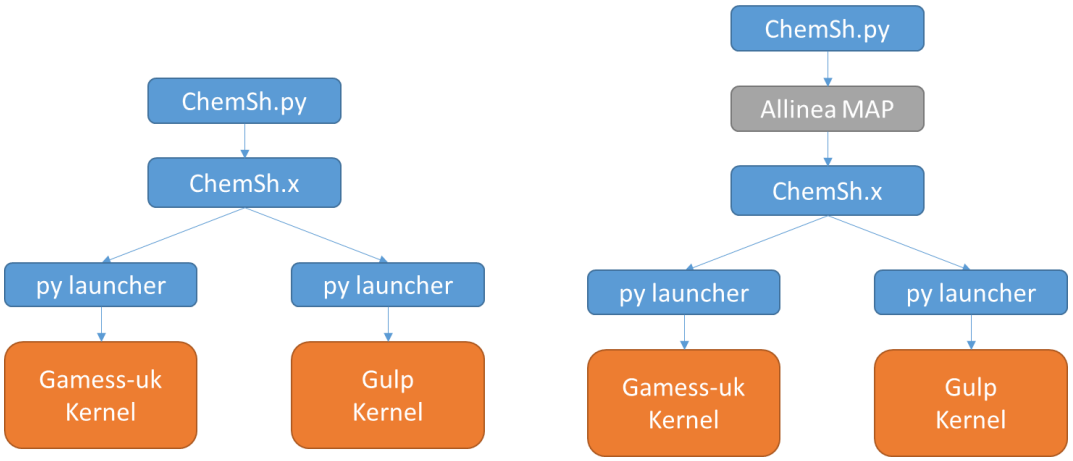

Figure 19: ChemShell framework and the interjection of Allinea MAP

Figure 19 shows the rough calling path of the ChemShell application making use of two kernels (GAMESS-UK and GULP). An initial python framework launches a 'chemsh.x' executable, written in C, this executable can be run with MPI and launces parallel kernels. However, the computational work actually takes place inside the specific kernels. This means to obtain a representative MAP profile we must ensure that the resulting kernels are profiled correctly.

Figure 19 shows where we have injected the Allinea MAP instrumentation, at the same level where MPI is used. This allows MAP to make use of the MPI environment and to track all processes which are spawned by the 'chemsh.x' application. As previously discussed the Allinea support for python is limited, which is a concern as python is being used to launch the kernels. However, in this case we are less concerned with the performance of the python component – only that of the kernels which are called.

```
`ALLINEA_MPI_INIT_PENDING=1 ALLINEA_MPI_INIT=MPI_Init map --profile`
```

Listing 1: ChemShell environment parameters for Allinea MAP

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

What is important is to be able to trace the call stack from the primary application into the execution kernels.

We note that because the MPI_Init is not called from directly within the chemsh.x application, but within one of the loaded libraries. The Allinea tools make use of the `MPI_Init` call as a synchronisation point, however having the call in a different library means that the function pointer is located in a different translation unit. This means that Allinea MAP may have issues identifying the correct location of the function call, which can inhibit profiling. As such, we have modified the chemsh.py python script to set a few environment variables to ensure the application is profiled correctly. This configuration is detailed in Listing 1**.**

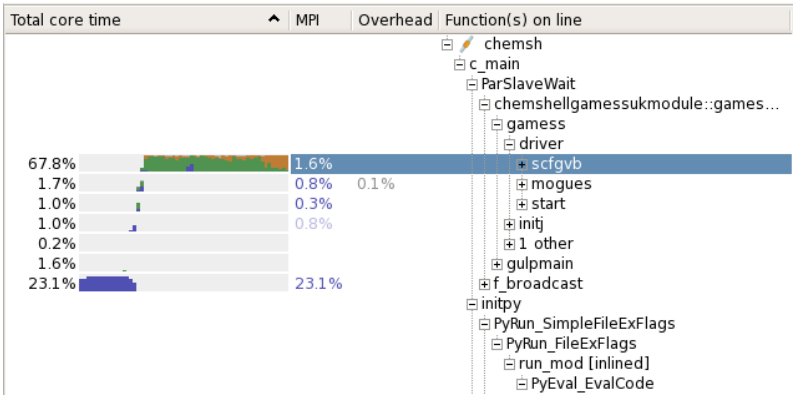

Figure 20: Call stacks within ChemShell

From Figure 20 we see an example of the generated call stacks for ChemShell within Allinea MAP. In this example we can see the top level calling application ‘chemsh’ with both a call to the ChemShell GAMESS-UK interface and the GULP main. Additionally, we can see activity within the python control. At this time, we only report the python function name, and not the line number or source code view.

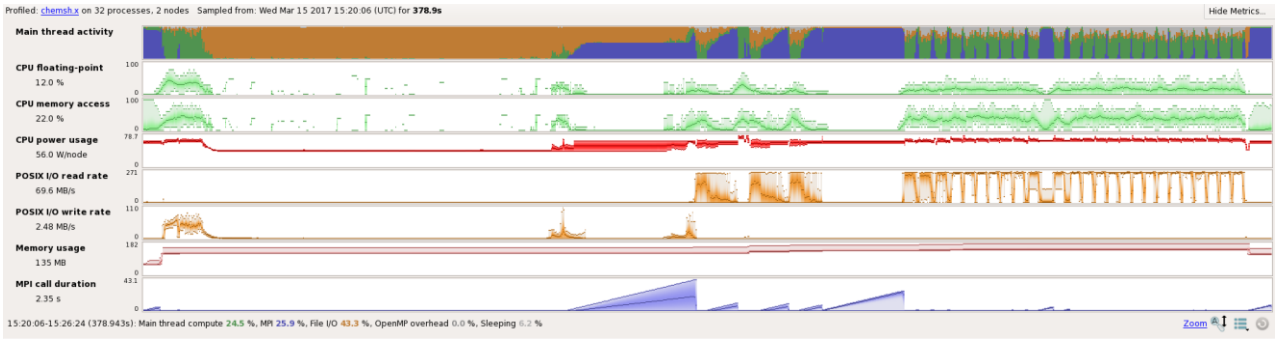

Figure 21: Profile of the ChemShell application in Allinea MAP

In the profile shown in Figure 21 we can see the ChemShell application running on the STFC platform Neale, across two nodes. The behaviour we see is heavily I/O dominated – in what appears to be an inefficient way. We see about 40% of the runtime performing I/O at the start of the job, which results in some load imbalance, which is indicated by the MPI call synchronisation time (acting as a barrier). The latter phase of the computations is composed of a high amount of CPU activity with some efficient data reads and some MPI communication – but minimal load imbalance.
Another interesting metric here is the energy consumption. We see it loosely follows the behaviour of the I/O profile – low energy during the inactive I/O phase – and drops during MPI activity.

4. MultiScale Visualisations

A key component for Work Package 4 was the development of a capability to visualise performance data from a multiscale simulation. Traditionally, the Allinea performance analysis tools (Allinea MAP and Performance Reports) capture data for the execution of a single application, and as such generate a single profile which can be visualised. Within the ComPat project, applications do not operate in a standalone manner, and different applications are more tightly coupled. In order to visualise data from interlinked applications, data needs to be collected and aggregated before presenting this back to the user. Using the existing GUI to visualise all of the performance data collected would provide too much detail and would be difficult to navigate.

4.1. Data Store

An important prerequisite of a data visualisation system is having a data store. Sections 2.2.2 and 2.3.2 detail the development of the new JSON export capabilities in the Allinea toolset. This allows for the data contained within individual profiles to be exported to a common format, which can be further processed or aggregated. As such, a data store targeted towards handling JSON would be most desirable.

Milestone 7 requires the generation of an ad-hoc capability to merge individual scale models to multiscale models. To these ends we have made use of the Elasticsearch [10] product , part of the Elastic cloud software stack. As a solution, Elasticsearch provides a very scalable means of storing and querying data, including in JSON format. An additional component in the Elastic software stack is Kibana. Kibana is a visualisation framework which sits on top of Elasticsearch. We present visualisations based on this software in Section 4.2.

4.2. Visualisation

Once the data from multiple executions is collected in an Elasticsearch store, we can start to form queries around the data to visualise it. Kibana is specifically designed to view temporal data, so we must express the queries in such a way that suits this representation. This is a limitation which will be addressed later in the project.

Figure 22 presents the first such multiscale visualisation, based on a 2048 core fusion application run. Specifically, we are plotting the time spent in MUSCLE2 receive calls for the four model kernels, simultaneously. What we see here is that the three, serial, auxiliary models are spending lots of time waiting for communications with the primary (Gem) kernel. We can see an obvious cyclic behaviour in the call duration, caused by the iterations of the calculation.

In Figure 23 we show how the visualisation framework can be used to compare runs of applications at different scale. In this scenario we focus on the CPU instruction mix of the Gem kernel from the fusion application when strong scaled across 512, 1024 and 2048 cores. From this visualisation we can see that the time being spent, as a percentage of instructions, in both memory (blue) and CPU floating point (green) decrease the more cores the application is run on. In this case the loss of CPU intensity is a result of increased MPI communications.
We can observe this by plotting the rate of MPI calls (calls / second) for the three different scale of the run. In Figure 24 we present a statistical breakdown of this call rate. As the Allinea tools collect information at a relatively high frequency we can begin to perform a statistical analysis on the data, in this case looking at the percentile (1^{st}, 25^{th}, 50^{th}, 75^{th} and 99^{th} percent) distribution of the call rate. From this we can observe that as we scale the core count not only does the mean call rate increase, but so does the variance. This suggests that the communication phases have become bunched, and that it alternates between periods of high intensity and periods of low intensity.

The more data that can be captured from the system the more scope there is for data visualisation and analysis. For example, multiple executions of the same application under similar conditions could be used to assess run-to-run performance reproducibility. Whereas multiple runs under different conditions could be used to assess sensitivity to environmental factors, such as node type.

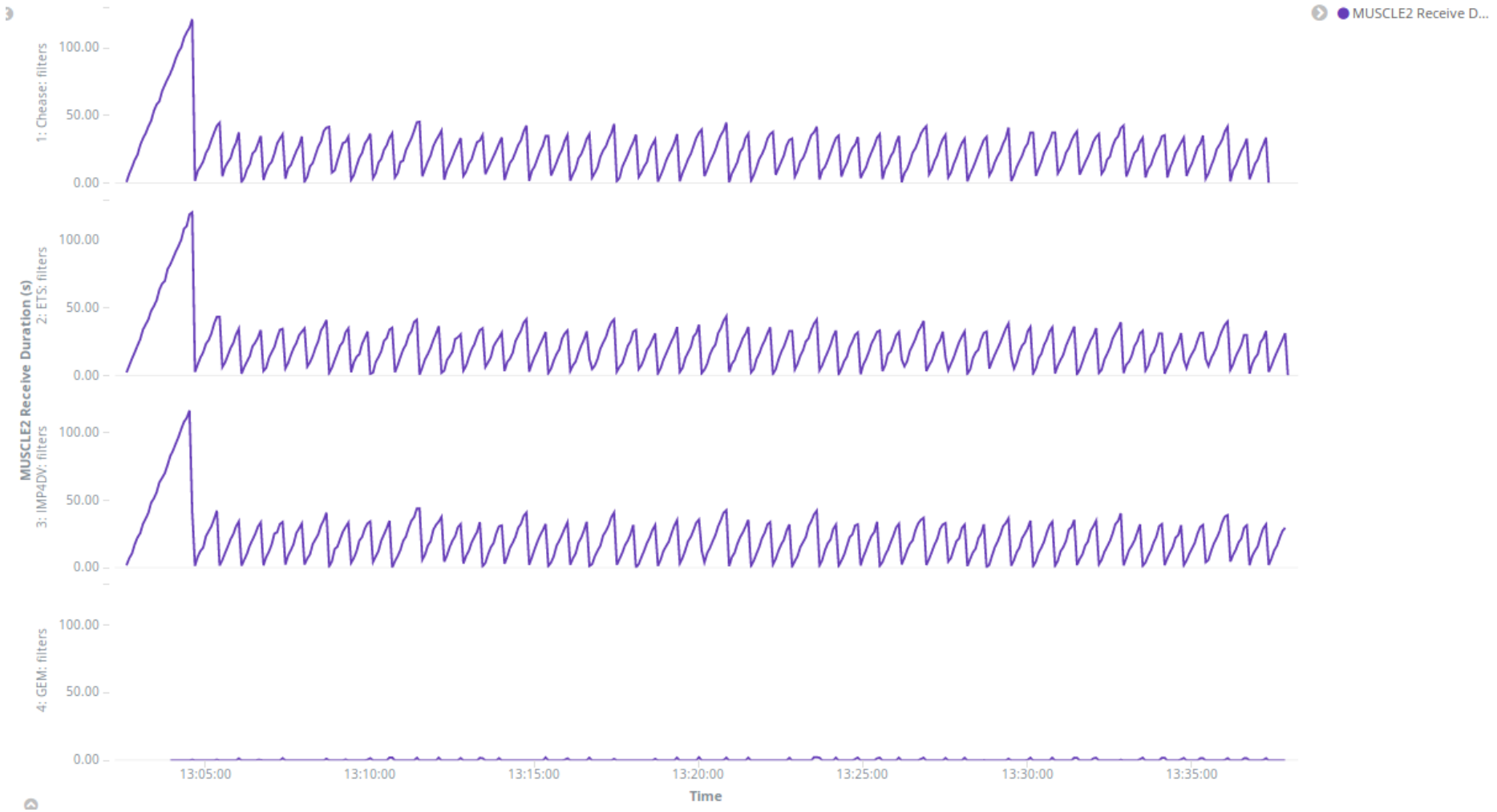

Figure 22: 2048 core Fusion multiscale simulation visualisation

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

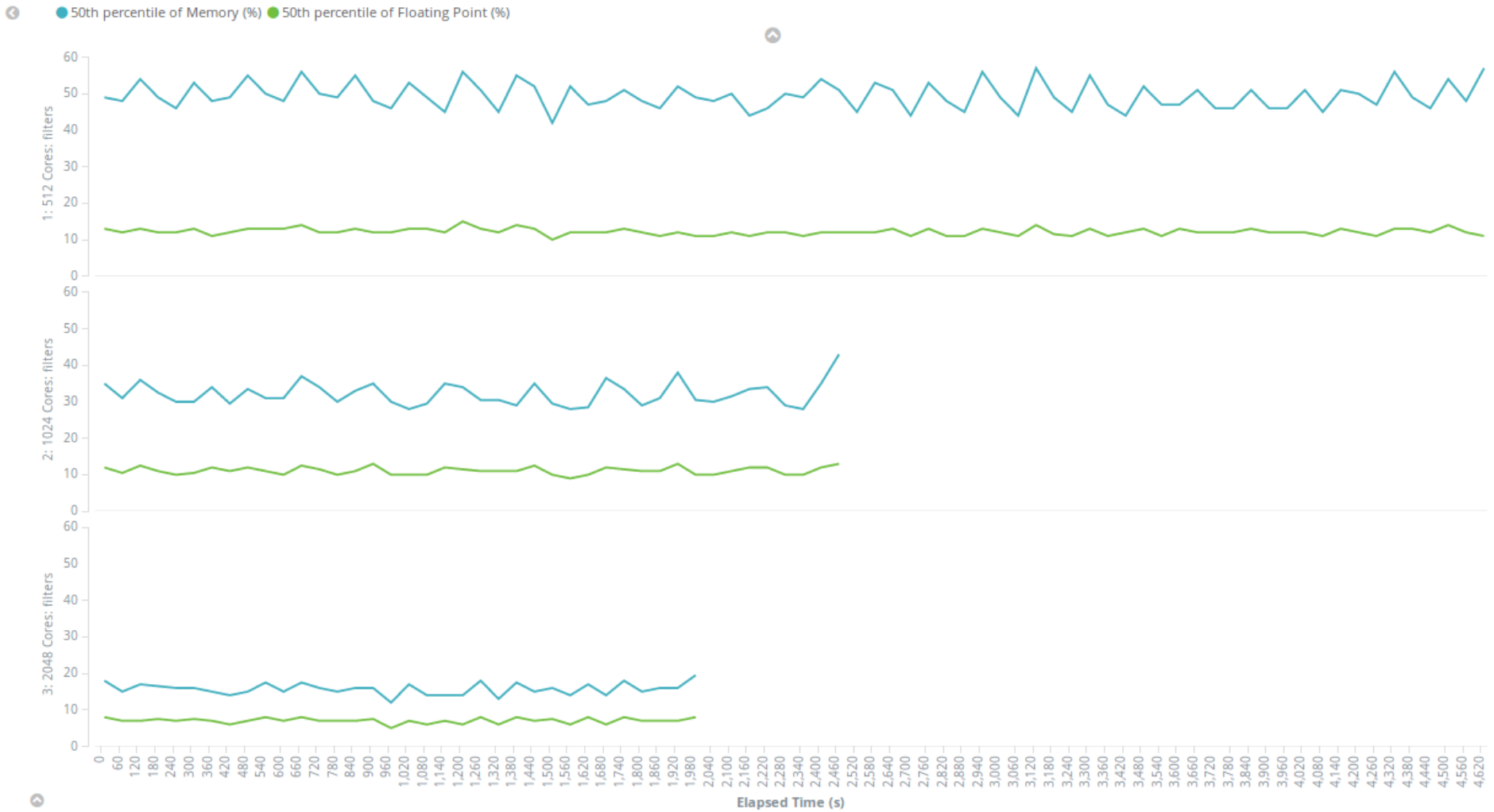

Figure 23: Scaling of instruction breakdown for Fusion application Gem kernel

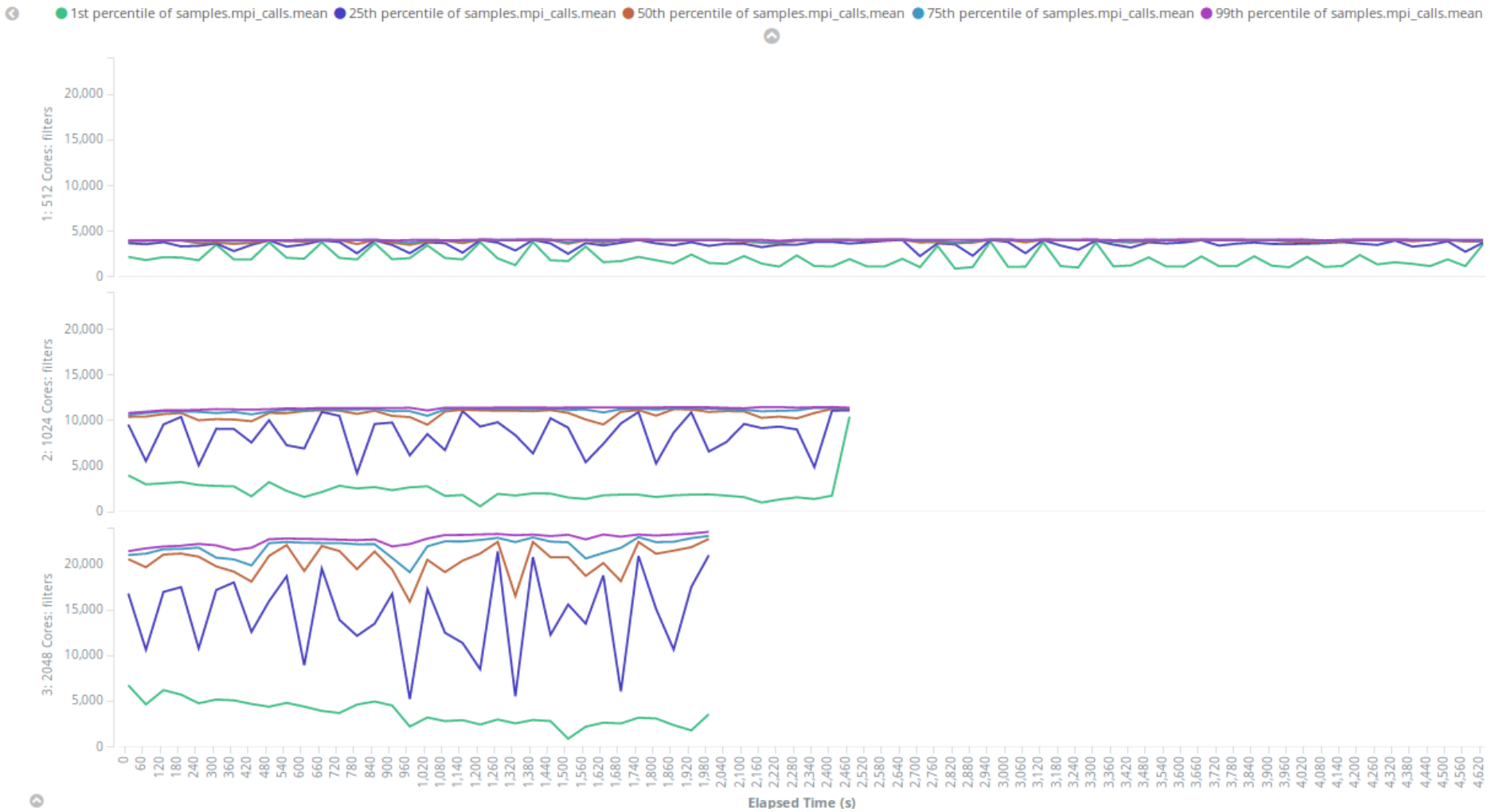

Figure 24: Distribution of MPI call rates for Figure 23 calculations

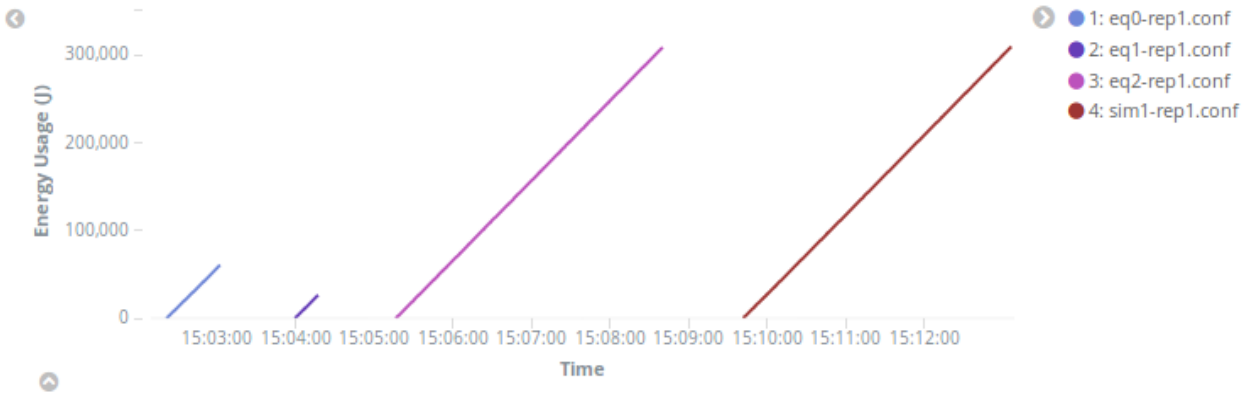

Figure 25: Energy usage profile of BAC simulation on 84 cores

In Figure 25 we show a BAC replica computing example, plotting energy usage of the four replicas over time. The gradient of these four lines represents the power draw across the nodes, and the fact that they are straight lines represents that there are no major power fluctuations within the application. Changing the gradient of these curves represents a change in the overall energy consumption of the simulation, which is important as core goal of the ComPat project to address energy efficient computing. Even by just collecting this information, on a kernel by kernel basis, we can feed the pattern service, and the performance matrix, to improve the capability to estimate energy usage and schedule accordingly.
We note that some of the time between the different application runs can be attributed to the overhead of the MAP profiler, incurred at the end of execution of generating the profiles.

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

5. Future Developments

5.1. Interactive Debugging

As discussed in Section 2.1 one of the current limitations of debugging with Allinea DDT is that there is minimal support for interactive debugging within the ComPat environment. As outlined in Deliverable 4.1, additional components are required to support the data forwarding from end nodes through QCG back to a GUI on the user's machine. This flow can be created with the QCG connect mode – a prerequisite of enabling this data flow was the implementation of a 'quiet mode' within QCG connect. This enables DDT tools to communicate through QCG with no additional data being injected by QCG, which would have caused difficulties within the DDT client. Now this mode is in place, work can begin to establish the additional forwarding client required.

Another issue faced has been the use of GSISSH on SuperMUC as an authentication wrapper to SSH. DDT normally sets up an SSH channel to communicate through, however the use of GSISSH has caused additional issues for the Allinea tools, preventing the channel from being established. It is hoped that by passing all communications through QCG we should have more control on the flow of data, and will be more successful in enabling interactive debugging.
Another issue faced has been the use of GSISSH on SuperMUC as an authentication wrapper to SSH. DDT normally sets up and SSH channel to communicate through, however the use of GSISSH has caused additional issues for the Allinea tools, preventing the channel from being established. It is hoped that by passing all communications through QCG we should have more control on the flow of data, and will be more successful in enabling interactive debugging.

5.2. Data Visualisations and Analytics

In Section 4 we presented the current state of multiscale visualisations, this represents an ad-hoc capability to merge and visualise the data from multiple components of a simulation. This capability is not mature, and more work needs to be undertaken to establish a formal, and automated, means to collect and visualise this data.

A key component of this will be working with the application teams in Work Package 3 and understanding their requirements for data visualisation. Additionally, it is important to undertake this work in close collaboration with Work Package 2, to ensure the full integration into the multiscale computing patterns and the overall ComPat workflow.

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

5.3. ComPat Stack Integration and Meta-Data

One of the really interesting research topics for this project is the level of integration with the pattern service and resource brokering, driven by the flow of information. By having the Allinea profilers run as part of multiscale simulations data can be captured at multiple points within the execution of these applications. However, alone this information lacks context from the wider execution environment. The computational pattern, and the MML description of the job, contain this context – providing information about exactly what was run, and in terms of the pattern why. This provides a valuable source of information to guide the multiscale simulations, and provide the explicit connectivity between individual activities.

Equally the flow of information must pass both ways, the performance data collected from profiling must go back into the patterns service to help annotate the computational submodels with information such as runtime and energy usage. This information can in turn feed the performance matrix of the pattern service – which can feed QCG with performance estimations. Closing this information flow loop is a key contribution to the ComPat environment.

6. Conclusion

In this deliverable, we have presented the progress on Tasks 4.2 and 4.3. We have shown the increased capability of the Allinea tools to integrate into the ComPat ecosystem and software stack, and have demonstrated a working profiling capability for four such applications. Work has initially focused on the profiling of applications, as opposed to debugging, due to the key importance of performance data within the project.

The primary focus for collection of new performance data has been the integration with the MUSCLE2 communication library. Specifically, we have presented the development of a custom performance metric to collect MUSCLE2 data from the communication library, and feed the data into Allinea MAP. On top of this we have developed a partial report plugin to display the MUSCLE2 performance data in Allinea Performance reports. The collection of this information, and other relevant metrics, can be used to feed back into the ComPat pattern service (WP 2), to improve the efficiency of the job submissions.

The development of support for the different multiscale applications has been documented in this report, highlighting the modifications made to enable profiling. Additionally, we have detailed any additional changes to enable the integration with the wider ComPat environment, specifically the QCG brokering package.

In addition to the collections of this extra data we have demonstrated the first capabilities to visualise multiscale performance data, by merging data from multiple individual application runs. This capability marks the completion of Milestone 7 (an ad-hoc capability for merging individual scale performance profiles to create one multiscale profile). Additionally, the software packages have been released and made available to project partners to constitute delivery of Milestone 8.2 (first version of higher level tools).

Annexes

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** [2] 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.

References

[1] Allinea Software Ltd., "Deliverable 4.1: Report and software on design of tools and required actions to support performance tools for multiscale," H2020 ComPat, 2016.

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

[3] A. Yoo, M. Jette and M. Grondona, “SLURM: Simple Linux Utility for Resource Management,” *Job Scheduling Strategies for Parallel Processing. JSSPP 2003. Lecture Notes in Computer Science, vol 2862.,* 2003.

[4] GitHub, “MUSCLE2,” [Online]. Available: https://github.com/psnc-apps/muscle2.

[5] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé and K. Schulten, “Scalable molecular dynamics with NAMD,” *J. Comput. Chem,* p. 26: 1781–1802, 2005.

[6] D. Case, T. Darden, T. C. S. Cheatham and J. e. a. Wang, “AMBER 12,” 2012.

[7] L. Kalé and S. Krishnan, “CHARM++: A Portable Concurrent Object Oriented System Based on C++,” *Proceedings of OOPSLA'93,* pp. 91-108, 1993.

[8] M. Guest, I. Bush, H. Van Dam, P. Sherwood, J. Thomas, J. Van Lenthe, R. Havenith and J. Kendrick, “The GAMESS-UK electronic structure package: algorithms, developments and applications,” *Molecular Physics,* vol. 103, 2005.

[9] J. D. Gale, “GULP: A computer program for the symmetry-adapted simulation of solids,” *J. Chem. Soc.,* vol. 93, no. 4, pp. 629-637, 1997.

[10] Elastic, “Elasticsearch,” [Online]. Available: http://www.elastic.co/.