SEVENTH FRAMEWORK PROGRAMME

FP7-ICT-2011-7

DEEP

Dynamical Exascale Entry Platform

Grant Agreement Number: 287530

D5.4
Performance analysis tools ready

Approved

Version: 2.0
Author(s): V. Beltran, BSC
Date: 16.01.2015
Performance analysis tools ready

Project and Deliverable Information Sheet

<table>
<thead>
<tr>
<th>DEEP Project</th>
<th>Project Ref. №:</th>
<th>287530</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Title:</td>
<td>Dynamical Exascale Entry Platform</td>
<td></td>
</tr>
<tr>
<td>Project Web Site:</td>
<td><a href="http://www.deep-project.eu">http://www.deep-project.eu</a></td>
<td></td>
</tr>
<tr>
<td>Deliverable ID:</td>
<td>D5.4</td>
<td></td>
</tr>
<tr>
<td>Deliverable Nature:</td>
<td>Report</td>
<td></td>
</tr>
<tr>
<td>Deliverable Level:</td>
<td>PU*</td>
<td></td>
</tr>
<tr>
<td>Contractual Date of Delivery:</td>
<td>31 / May / 2014</td>
<td></td>
</tr>
<tr>
<td>Actual Date of Delivery:</td>
<td>31 / May / 2014</td>
<td></td>
</tr>
<tr>
<td>EC Project Officer:</td>
<td>Luis Carlos Busquets Pérez</td>
<td></td>
</tr>
</tbody>
</table>

* - The dissemination level are indicated as follows: PU – Public, PP – Restricted to other participants (including the Commission Services), RE – Restricted to a group specified by the consortium (including the Commission Services). CO – Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

<table>
<thead>
<tr>
<th>Document</th>
<th>Title: Performance analysis tools ready</th>
<th>ID: D5.4</th>
<th>Version: 1.0</th>
<th>Status: Approved</th>
<th>Available at:</th>
<th>Software Tool: Microsoft Word</th>
<th>File(s): DEEP_D5.4_Performance_analysis_tools_ready_v2.0_ECapproved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorship</td>
<td>Written by:</td>
<td>V. Beltran, BSC</td>
<td>Contributors:</td>
<td>J. Gimenez (BSC), G.Llort (BSC), M.Knobloch (JUELICH)</td>
<td>Reviewed by:</td>
<td>P.Arts (Eurotech), N.Eicker (JUELICH)</td>
<td>Approved by:</td>
</tr>
</tbody>
</table>

Document Status Sheet

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>31/May/2014</td>
<td>Final</td>
<td>EC submission</td>
</tr>
<tr>
<td>2.0</td>
<td>16/January/2015</td>
<td>Final</td>
<td>EC approved</td>
</tr>
</tbody>
</table>
Document Keywords

| Keywords: | DEEP, HPC, Performance Analysis Tools, Instrumentation, Visualisation, Analysis, Paraver, Extrae, Scalasca, Exascale |

Copyright notices

© 2011 -2015 DEEP Consortium Partners. All rights reserved. This document is a project document of the DEEP project. All contents are reserved by default and may not be disclosed to third parties without the written consent of the DEEP partners, except as mandated by the European Commission contract 287530 for reviewing and dissemination purposes. All trademarks and other rights on third party products mentioned in this document are acknowledged as own by the respective holders.
Table of Contents

Project and Deliverable Information Sheet .................................................................i
Document Control Sheet ............................................................................................i
Document Status Sheet .............................................................................................i
Document Keywords ..................................................................................................ii
Table of Contents ......................................................................................................iii
List of Figures ...........................................................................................................iii
Executive Summary ....................................................................................................1

1 Introduction .............................................................................................................2
2 DEEP Offload .........................................................................................................2
3 Paraver ....................................................................................................................3
  3.1 Port of Extrae to Xeon Phi ..................................................................................3
  3.2 Extension of Extrae to support the DEEP Offload ...........................................4
    Instrumentation phase .........................................................................................5
    Merging phase ....................................................................................................6
  3.3 Use case example .............................................................................................8
4 Scalasca ..................................................................................................................10
  4.1 Port to Xeon Phi ..............................................................................................11
  4.2 Support for (node-level) OmpSs ......................................................................12
  4.3 Extensions to Cube GUI ..................................................................................13
    4.3.1 Flexible system tree ..................................................................................13
    4.3.2 Cube derived metrics ...............................................................................14
  4.4 Support of DEEP Offload ...............................................................................15

References and Applicable Documents .......................................................................17
List of Acronyms and Abbreviations ..........................................................................18

List of Figures

Figure 1: Trace file of a Cholesky phase using OmpSs. .............................................4
Figure 2: Contents of the TRACE[-n].spawn file ..................................................6
Figure 3: Example of TRACE[-n].spawn files. Rank 0 of master application spawns a second application with 2 MPI tasks. (a) TRACE.spawn (master); (b) TRACE-2.spawn (spawned) ..........6
Figure 4: Thread 1.1.1 calls MPI_Spawn_multiple (deep_booster_alloc) to start 4 children (gray). ....7
Figure 5: Communication matching process ..........................................................8
Figure 6: Timeline showing the first iterations of the elastic standalone FWI algorithm. Colors represent the execution of the different user tasks ...........................................9
Figure 7: 1-step of the preconditioner and gradient computation phase of 4 data sets in parallel ..........9
Figure 8: 1-step of the preconditioner and gradient computation phase of a single data set................10
Figure 9: OmpSs internal communications between parent and children processes. ..................10
Figure 10: Cube screenshot of NAS BT benchmark run on Stampede. ......................11
Figure 11: Cube screenshot of OmpSs run ................................................................13
Figure 12: Example of Cubes Flexible System Tree ..................................................14
Figure 13: Derived metric creation window. ................................................................15
Executive Summary

The DEEP project intends to develop a novel supercomputer architecture that radically departs from existing approaches. While traditional supercomputers are homogeneous at the cluster level but can be heterogeneous inside each node by using some kind of accelerator, our approach targets a heterogeneous supercomputer composed of two interconnected clusters of Cluster Nodes (CN) and Booster Nodes (BN), both of them homogeneous. The CNs are traditional compute nodes that can collectively offload computational kernels to the BNs, which form a cluster of accelerators. The flexible nature of the DEEP concept challenges the traditional software stack: applications architecture, programming models, job managers as well as performance analysis tools.

This deliverable describes the analysis tools developed and extended in the context of the DEEP project to identify the different computational phases of an application to study then the performance and scalability of each phase. Good analysis tools are necessary to achieve optimal performance on any high performance system. However, the novel Cluster-Booster architecture proposed on the DEEP project requires an even deeper understanding of the application characteristics and phases to be able to make the most of the heterogeneous nature of the system. On DEEP, it is mandatory to appropriately identify the application’s phases that will run better on either the Cluster or the Booster side to achieve optimal performance. Hence, appropriate analysis tools to choose the best Cluster/Booster partition are necessary. Once a given Cluster/Booster partition of the application has been decided, powerful analysis tools are again mandatory to validate the suitability of such application’s phase partition and mapping.

Paraver/Extrae and Scalasca are two powerful and well-known analysis tools for HPC environments, which already work out-of-the-box for the Cluster side of the DEEP System. Thus, they can be efficiently used to identify the different computational parts of each application. However, neither of these tools previously supported the challenging architecture presented in DEEP. This deliverable explains these challenges to analyse applications that run on the DEEP architecture as well as the solutions implemented in Paraver/Extrae and Scalasca.
1 Introduction

Performance analysis tools are fundamental to study, analyse and understand performance and scalability of HPC applications. The increasing complexity of current computing systems makes it very hard to predict the performance of any application on a new system. Thus, good analysis tools are mandatory to analyse and understand application bottlenecks before any optimisation can be done. Any application optimisation should be driven by the outcome of a thorough analysis since otherwise, a lot of time and effort can be wasted addressing minor problems or issues that do not even exist. On the DEEP architecture the availability of powerful analysis tools is even more important because the performance characteristics of its novel architecture are not well known yet. The rest of this deliverable is divided in two main sections that cover Paraver and Scalasca analysis tools respectively. The purpose of these two analysis toolboxes is twofold: in a first step, these tools will be used to analyse the behaviour of WP8 applications on a regular HPC cluster. From this analysis, application developers will gain insight on the computational structure of their application to decide an initial Cluster/Booster partition of the code. Once this partition is implemented using the DEEP offloading model presented in D5.3, the analysis tools will be used again to validate the suitability of the chosen partition on the DEEP System. Thus, it is important to extend both Paraver and Scalasca to support DEEP applications as a whole, including the Cluster part, the Booster part and the interactions between both.

The rest of the deliverable first recaps how the DEEP offloading works. Then the work done on Paraver and Scalasca is presented on Section 3 and Section 4, respectively.

2 DEEP Offload

The DEEP programming model, presented in D5.1 and D5.3, is a hybrid programming model that extends the well-known MPI+OmpSs/OpenMP model with specific extensions developed to ease the offload of MPI-kernels from CNs to BNs. The offload of high performance kernels written in MPI is a key feature to effectively exploit the DEEP architecture. Current MPI implementations already support the MPI_Comm_spawn() primitive to dynamically create new MPI processes on arbitrary nodes. This collective operation creates an inter-communicator that connects the set of nodes that call it, with the new set of MPI processes created on the specified remote nodes. It is worth noting that this MPI primitive supports the creation of new MPI processes even on nodes with a different architecture, which makes it especially well-suited for the DEEP Architecture. However, this function can dramatically increase the complexity of a MPI application since the programmer has to coordinate and manage two or more sets of parallel MPI processes and to explicitly send the required data from side to side. This additional complexity makes this technique cumbersome or even unfeasible on large and complex applications such as the ones targeted by DEEP.

To overcome the above-mentioned issues, OmpSs has been extended to support a flexible approach to offload MPI kernels from CNs to BNs. The goal is to ease, as much as possible, the mechanism to offload MPI kernels, but leveraging the current MPI infrastructure. To that end, we have designed an offload model that uses #pragmas to mark the MPI codes that must be offloaded. The compiler and the runtime systems cooperate to transparently manage all the data transfers between the MPI processes on the Cluster and the Booster. This pragmatic approach leverages both the unmodified MPI kernels of the current application and the optimised communication infrastructure provided by the MPI library. The runtime uses, under the hood, the MPI_Comm_spawn(), MPI_Send(), etc., primitives to setup and transfer the required data from CNs to BNs and vice-versa. This approach increases the malleability of the
applications and has the potential to apply advanced optimisations using our user-transparent
directory/cache implemented on the Nanos++ [1] runtime system. D5.3 contains further
details and several examples on the DEEP offload.

The key step to analyse an application that uses the DEEP offloading is to properly instrument
the \texttt{MPI\_Comm\_spawn()} call. This is not an easy task since it involves the synchronisation
and merge of different sets of events produced on different nodes. However, if this
instrumentation is done correctly, it will support not only the DEEP offload but also any
native MPI applications that directly calls \texttt{MPI\_Comm\_spawn()}.

## 3 Paraver

Paraver [2] is a very flexible data browser that allows displaying a huge number of
performance metrics and provides precise details about the application behavior and the
resources used. An application can have one or several tasks, and each task can have one or
several threads. Paraver defines a process model to represent these different objects in the
trace. The mapping of the MPI programming model onto the Paraver process model can be
done as follows:

- Each MPI process is a Paraver Task with one or several Paraver Thread.
- The whole MPI application is the Paraver Application, grouping all MPI processes.
- Multiple MPI applications might be grouped in a Paraver Workload.

This mapping leads to have the execution of an MPI application and all its spawned MPI
processes in a single trace at the same time. Thanks to Paraver's flexible approach, no changes
were required to be made to this tool.

Extrae [3] is a powerful instrumentation library that is seamlessly integrated with Nanos++
and MPI. Extrae generates a trace file with useful information about the application’s
execution, which can be analysed with the Paraver visualisation tools to look for bottlenecks
or other issues. All the work required to support the Xeon Phi platform and the DEEP offload
has been implemented on Extrae.

### 3.1 Port of Extrae to Xeon Phi

To validate the port of the OmpSs programming model to the Xeon Phi the Extrae
instrumentation library was also ported to this platform. While Xeon and Xeon Phi have
different architectures they share a similar compilation tool-chain. Hence, the main work to
port Extrae to the Xeon Phi platform was to port several libraries (libxml, libz, PAPI, etc) that
required some modifications to run on this platform. Extrae was used then to collect traces of
all benchmarks and applications analysed on D5.1. Figure 1 shows as an example a trace of a
Cholesky benchmark running on a Xeon Phi with 48 hardware threads.
3.2 Extension of Extrae to support the DEEP Offload

In a second step Extrae has been extended to support the unified instrumentation and visualisation of offloaded applications that run one part on the CNs and on the BNs respectively. Both parts of the program are instrumented like a regular MPI application with the difference that when the deep_booster_alloc() [4] function is called, additional information is recorded into the trace to be able to synchronise the MPI processes on the CNs with the new MPI processes spawned on the BNs. Nanos++ also emits events for all the relevant actions between CNs and BNs (e.g. start of task offloading, end of task offloading, copy of inputs from CN to BN, copy of outputs from BN to CN, etc.). At the end of the execution, a final merging step is performed to generate a single trace of the whole program execution. This trace contains all the information from all CN and BN processes and the communications between them. Later on it can be visualised and analysed on Paraver.

3.2.1 Implementation details

The current development starts from Extrae, a tracing toolkit that collects automatic performance measurements from most parallel programming models, including MPI, OpenMP and OmpSs. For the current discussion only the MPI support is relevant. Extrae provides three mechanisms to instrument MPI applications: 1) dynamic binary instrumentation based on the code-patching library Dyninst [3]; 2) preloading of shared libraries; and 3) static override of the MPI standard profiling interface (PMPI). Once attached to the application through any of these three interposition methods, Extrae automatically collects performance measurements at the entry and exit points of most calls to the MPI runtime. The information gathered (timestamp, hardware counters and call stack information) is stored into separate files for each parallel task and, after the execution, all files are merged.
into a single Paraver trace. In order to support the tracing of spawned processes, new features were implemented both in the instrumentation and the merging phases as detailed below:

**Instrumentation phase**

1. When a program uses DEEP offloading, the OmpSs runtime ends up invoking `MPI_Comm_spawn()` or `MPI_Comm_spawn_multiple()` to deploy new processes to the Booster Nodes. We have included new wrappers into the Extrae runtime, with interfaces for both C and Fortran codes, in order to intercept the calls to these MPI routines. To do so, it is necessary to redefine both symbols in the Extrae tracing library with the same prototype as in the real MPI implementation. In this way, Extrae is able to take control of the execution when OmpSs starts an offload. Once intercepted, the first action done by Extrae is to call the real implementation of the symbol (defined in the MPI library) in order to spawn the processes. On the parent-side, this operation returns a new MPI inter-communicator that can be used to communicate with the children processes. On the children-side, the existing wrappers for the initialisation routines `MPI_Init()` and `MPI_Init_thread()` have been extended so as to retrieve on each child process its local identifier of the inter-communicator to their parents, by calling to `PMPI_Comm_get_parent()`. Right after the new processes are created and initialised, Extrae uses the new inter-communicators on both ends to exchange information between all parents and children. This information regards to the ranks of the processes involved in the spawn operation, the time when the spawn was performed, and the local identifier that every task has for the new inter-communicator. In particular, this information is broadcasted from the root rank on the parent-side of the spawn collective to all children, and from the rank 0 child to all parents, through calls to `PMPI_Bcast` on the new inter-communicator. In this way, all processes get to know each other, which will be necessary to be able to correctly trace who is the partner in any following peer-to-peer communication between parents and children.

2. The information about the ranks, communicators and timestamps that is exchanged between parents and children during the spawn is written to a file to be used later on the merging phase. The new output file is named `TRACE[-n].spawn`, where `n` is a numerical identifier starting from 2 for each new MPI kernel spawned (the master application does not include any suffix to preserve backwards compatibility with the tool). In order to support multiple tasks calling the spawning routines concurrently, this identifier must be unique for each application. To select the identifier, the rank 0 process of each application tries to catch an advisory lock on this file through the `flock()` system call during the initialisation. On failure, the identifier is increased until the lock is successfully taken. This mechanism assumes that all processes will have access to the same directory, which in our case is granted by the GPFS filesystem. The contents of this file are shown in Figure 2. The first line contains the time delta from the parent’s `MPI_Init()` to the `MPI_Comm_spawn()`, which will be used in the merging phase to synchronise the events of the children processes. The following lines specify pairs of local ranks and communicator identifiers that connect to a spawned MPI kernel, which will be also used in the merging phase to correctly match the communications between partners that belong to different MPI kernels.

Figure 3 shows an example of the generation of these files. In this example, only the rank 0 process of the master application (a) spawns 2 MPI processes (b). All communications from the parent’s rank 0 through the communicator 7588480 target the MPI kernel 2. On reverse direction, all communications from the child process 0 through communicator
7474560, or from process 1 through communicator 7474096, target the master application 1.

```
<spawn-time-offset>
<rank>  <local-comm-id>  <target-app>
<rank>  <local-comm-id>  <target-app>

...  
```

Figure 2: Contents of the TRACE[-n].spawn file

```
0   7588480   2
0   7474560   1
1   7474096   1
10013082559
```

Figure 3: Example of TRACE[-n].spawn files. Rank 0 of master application spawns a second application with 2 MPI tasks. (a) TRACE.spawn (master); (b) TRACE-2.spawn (spawned)

3. All peer-to-peer messages going through an inter-communicator (between parents and children) to a specific target rank are also intercepted, and the target identifier stored in the trace is replaced by the global rank of that process in its MPI_COMM_WORLD communicator. This modification is required for the merge phase to be able to correctly distinguish the target rank of the messages between spawned processes. This is because when messages are sent from children to parents, from the child perspective, the parent ranks are relative to the parent communicator doing the collective spawn. For example, if every parent rank makes an independent spawn, all children processes would see a single parent process with rank 0, and that information would not be enough to distinguish the correct target of the message.

4. Since the service threads of the OmpSs runtime may initiate MPI communications concurrently with the user application, it is required that Extrae supports the tracing of multithreaded MPI. In order to add compatibility for multithreading, several internal structures that store temporary information of ongoing communications were protected with locks so that multiple threads can access them safely.

**Merging phase**

1. The merging tool mpi2prv unifies all execution traces from the different MPI tasks produced by Extrae into a single Paraver trace. This tool also provides support for unifying multiple applications' trace files, and so the merging of the traces of different spawned MPI kernels has been built on top of this functionality that already existed. Invoking the tool with the argument “--”, the user can specify a list of the different trace-files to merge.

2. One of the tasks performed during the merge phase consists in synchronising the timestamps. Since the spawned processes can run on nodes different from those of the parent application, and these nodes may be governed by different clocks, it is necessary to implement a mechanism to synchronise the timestamps of the different applications. To do so, the mpi2prv uses the time delta that is stored during the execution (refer back to Figure 2) when the spawning calls are invoked to shift the timestamps of the traced events. More specifically, each spawned MPI kernel annotates the time that passed in its parent process from the `MPI_Init()` until the spawning call, and this offset is subtracted from the timestamps recorded for the events of the spawned processes. This results in
synchronising the parents’ spawning call with their children's MPI_Init, as shown in Figure 4. By the end of the master's spawn, all 4 children synchronise in the MPI_Init (green). After this, the master sends 1 task to each child to perform a second-level spawn of 4 children each. Second-level spawns are serialised due to limitations in the current implementation of the Intel MPI library.

Figure 4: Thread 1.1.1 calls MPI_Spawn_multiple (deep_booster_alloc) to start 4 children (gray).

3. Another task that was affected by these changes is the communication matching process. This process consists in correlating all pairs of peer-to-peer send and receive operations, in order to write a communication record in the Paraver trace that connects the partners, which is displayed in the timeline as a yellow line between both tasks, as shown in Figure 5. The existing algorithm determines the matching between sends and receives by the messages' tag and size, but did not allow making matches between different applications. The reason for this restriction is that the merging of multiple applications was first implemented to unify the traces of completely independent binaries, and so there were no communications between applications involved. Now this restriction has been removed, using the information about the processes involved in the spawn operation and the inter-communicators identifiers that are registered during the execution to build a translation table that enables the mpi2prv to correctly identify the target application of a given message. In Figure 5 yellow lines show communications happening between parent and children processes. The first 4 waves of communications correspond to OmpSs initialisation messages. After every process has been spawned (white means not yet created), the following communications are control and data messages for OmpSs task offload. The remaining communications correspond to the user-level MPI calls. These different types of communications can be filtered and displayed separately.
3.3 Use case example

Full Waveform Inversion is a cutting edge technique that aims towards the acquisition of the physical properties of the subsoil. Starting from a guessing (initial model) of the variables being inverted (e.g., sound transmission velocity), the stimulus introduced and the recorded signals, Full Waveform Inversion performs several phases of iterative computations to reach the real value of the variable set being inverted under an acceptable error threshold.

In this example, we evaluate a proxy benchmark that uses two nested levels of DEEP-offloading to replicate the behaviour of the production-class code. The real application first distributes the data sets among different nodes, and then on each node, a MPI application is run to process each of the data sets in parallel. In the proxy, the first offload corresponds to the data set distribution, and the second offload corresponds to the domain decomposition processing of each data set. In this case, we prepared a workload that required 4 domains per data set. Overall, this execution requires a total amount of 17 tasks (1 for the master task that offloads 4 children to distribute the work, and each child spawns 4 more to process the data set). Figure 6 shows this structure: thread 1 (row 1) performs the first offload, which creates 4 workers to distribute the data (light green). These workers decompose the data set (dark green and yellow) and offload this task to their workers, which perform the computations (dark blue). It is easy to observe that there is two types of computations (longer and shorter), which correspond to two different phases of the algorithm: the preconditioner and gradient computation phase (long) and the optimisation phase (short). As the reader can see, the seamless integration of Extrae and OmpSs allows us to capture the different tasks that the OmpSs runtime and the application processes are performing. Moreover, the mechanisms developed to instrument the spawned processes have enabled us to present the information of all the processes involved in a single, unified view, where the user can easily identify the different phases of the execution.
Figure 6: Timeline showing the first iterations of the elastic standalone FWI algorithm. Colors represent the execution of the different user tasks.

Figure 7 shows 1 step of the preconditioner and gradient computation phase of 4 data sets performed by the 16 second-level offloaded workers. Figure 8 shows a detailed zoom of this phase for a single data set. The colours represent different phases of the computation. Yellow lines represent MPI communications between these processes, where the user can easily identify a clear pattern of communications that are performed in the exchange_boundaries phase (white).

Figure 7: 1-step of the preconditioner and gradient computation phase of 4 data sets in parallel
Performance analysis tools ready

Figure 8: 1-step of the preconditioner and gradient computation phase of a single data set.

Figure 9 shows the same time region, but now the lines represent the internal communications of the OmpSs runtime to offload the tasks. The modifications implemented in Extrae have enabled us to display the communications that are internal within a process, as well as those happening between parent and spawned processes.

Figure 9: OmpSs internal communications between parent and children processes.

4 Scalasca

Scalasca [5] is an open-source tool-set for performance measurement and analysis of large-scale parallel programs. It supports MPI, OpenMP and hybrid MPI/OpenMP applications written in C, C++ or Fortran. It supports both call-path-based profiling as well as application tracing, making it a versatile tool for the whole process of performance analysis.

When profiling, performance data is aggregated at runtime, so it is for example possible to see how much time was spend in a function, but not distinguish the individual calls to that function. When tracing, events are created at specific points in the application run, e.g. at every function enter/exit, MPI messages, etc., so each function call can be analysed independently. However, tracing has in general a much higher overhead than profiling.

Scalasca comes with its own graphical user interface called Cube, which is used to visualise performance reports from both profiling and tracing, providing a single interface to the user to interact with. Cube also offers an interface to interact with other tools, e.g. show the worst instance of wait-state detected by the Scalasca trace analyser in Paraver.
Score-P [6], a common development of several performance analysis tools developers in Germany and beyond, is now used as a measurement system and profiler. It further generates OTF2 trace files, which can be analysed automatically by Scalasca or manually with Vampir, a time-line analysis tool similar to Paraver (see Section 3).

All new feature developments for measurement and profiling of the tool groups involved in the Score-P community, including the ones in the DEEP project, are now done for Score-P instead of the former respective tools. This makes these features usable by multiple tools and reduces duplicate work.

4.1 Port to Xeon Phi

As the Intel Xeon Phi is essential for the DEEP architecture, tools support for this platform is of paramount importance for the project. Fortunately, the software environment is pretty similar to traditional CPUs, so the porting of Scalasca involved mostly the configure and build system, i.e. the Xeon Phi has to be detected and the corresponding compiler and linker flags have to be added. We only added support for native mode, i.e. the whole application is running on the Xeon Phi, and for the symmetric mode, i.e. host CPUs and Xeon Phi running the same MPI application. We don’t support Intel’s offload mechanism, as it is not necessary for the project and probably soon replaced by OpenMP 4.

Figure 10 shows a screenshot of a NAS BT benchmark running on Stampede, using two MPI processes on the Xeon nodes and 15 MPI processes on Xeon Phi.

![Figure 10: Cube screenshot of NAS BT benchmark run on Stampede.](image-url)
4.2 Support for (node-level) OmpSs

As detailed in Section 2, the proposed programming model for the DEEP System is hybrid MPI + OmpSs/OpenMP plus the offload via MPI_Comm_spawn, so the performance analysis tools have to support that programming paradigms. So far, Score-P only supported hybrid MPI/OpenMP, but not OmpSs. We added support for node-level OmpSs directives in Score-P. This involved three components, a new threading subsystem, a Score-P OmpSs adapter and an instrumentation plugin based on the OmpSs API.

To support general threading paradigms other than OpenMP, we introduced a generic threading extension to Score-P, allowing for measurement of arbitrary threading models via subsystems. The OmpSs subsystem is based on Pthreads and manages the thread-local information when dealing with OmpSs tasks. It further supports hybrid MPI/OmpSs measurements, which is a requirement for DEEP.

For the connection between the OmpSs runtime and the Score-P measurement system an instrumentation plugin has been created using an API provided by OmpSs. This plugin identifies and translates OmpSs events for the Score-P adapter, which includes filtering of unneeded events, creating matching begin and end events and extracting additional information like task names and call sites. This plugin also introduces an abstraction layer between the OmpSs and Score-P runtimes, allowing for an independent building process with minimal dependencies.

The Score-P adapter integrates these events into the Score-P task and region model and manages the thread handling by applying the recently introduced generic threading model. As part of the Score-P measurement system the OmpSs events can now be recorded together with other event sources within Score-P, e.g., events generated by compiler or MPI instrumentation.

As OmpSs tasks are executed asynchronously, they cannot be mapped directly in the call tree. Cube visualises OmpSs tasks by using the representation of the generic task model used in Score-P, e.g., for OpenMP tasks. For this, an additional root node has been introduced as a sibling of main for the display of the task executions as can be seen in Figure 11. The task model in its current state shows the full call tree of a task below the task_root node while for every execution point of this task stub nodes are introduced into the calling call path. Currently, the matching between execution stub node and task_root call tree has to be done by matching names manually. For the two task definition modes, as task function or an outlined task region, different naming schemes are used. The explicit task functions are named by function name and call site with file name and line number. The outlined tasks are automatically generated and have the parent function and a number as names and contain the same call site information as the explicit tasks. The OmpSs adapter now supplies all events and regions for OmpSs runtime functions, like creating, scheduling and executing tasks, synchronisation and task waits.
4.3 Extensions to Cube GUI

In the course of the project it became necessary to extend Cube, the Graphical User Interface of Scalasca, in order to deal with the expected dynamical behavior of applications on the DEEP System. We identified two major enhancements, the flexible system tree and derived metrics.

4.3.1 Flexible system tree

Cube shows the performance properties of an application in three panes. The left pane (metrics pane) shows the different performance metrics. The middle pane (call tree) shows how these metrics are distributed among the call tree of the application. Finally, the rightmost pane (system tree) shows how the selected metric on the selected node of the call tree is distributed among the processes or threads on the hardware. So far, this pane showed a static layout of the nodes of a system with the processes and threads. The DEEP programming model is more dynamic. It might happen that the number of MPI processes on a booster node changes for different offloaded code parts, which cannot be handled by the current system view. Thus, we extended this pane to support a very flexible structure, which can be used on traditional systems to show for example node-level hardware counters, see Figure 12 for an example, as well as for the DEEP model, where an offloaded code part can create a new node in the system tree.
4.3.2 Cube derived metrics

Unlike a visual trace analysis tool like Paraver, Cube shows the data aggregated over the whole application run, which allows for a compact representation of the data, but loses the time dimension. Further, Cube presented the metrics individually. It was possible to select multiple metrics to aggregate their values, but that only made sense for metrics of the same kind, like time for MPI calls and time for I/O. To help the developers to understand the application behavior better, we introduced derived metrics, which allows correlation of arbitrary metrics. To do that, a Cube internal programming language, called CubePL, was developed. Figure 13 shows a screenshot of the "create derived metric"-window of Cube. There we create an average time per visit metric. Derived metrics still work on aggregated data, so the resulting metrics are in general averages if divisions occur in the CubePL statement.
4.4 Support of DEEP Offload

Full support for the DEEP offload mechanism is still under development for the Score-P measurement system. We expect to have a fully functional prototype, i.e. profiling, trace file collection and automatic trace analysis are working, when production starts on the DEEP System in December.

Especially the collection of application traces and the automatic trace analysis require substantial changes and enhancements in multiple parts of the measurement system and analyser, as well as new records in the trace file format. For the automatic replay in the Scalasca trace analyser we require more contextual information on resources and MPI communicators than time-line visualisation tools like Paraver and we currently investigate how to obtain these information with acceptable overhead. Further, other than Extrae (as detailed in Section 3.2), Score-P merges trace files at runtime (currently in MPI_Finalize), so this has to be changed to a dynamic merging of traces at MPI_Comm_disconnect.
However, profiling support requires only minor additions to Score-P. That helps understanding the application behaviour on the DEEP System and supplements the analysis capabilities of Paraver (as described in Section 3.3). We plan to have a working implementation by June to support the application developers in their first steps on the DEEP prototype platform.
References and Applicable Documents

## List of Acronyms and Abbreviations

### A

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>Booster Node (functional entity)</td>
</tr>
<tr>
<td>Booster</td>
<td>Hardware subsystem of DEEP comprising all Booster Nodes and intra-Booster network</td>
</tr>
</tbody>
</table>

### C

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN</td>
<td>Cluster Node (functional entity)</td>
</tr>
</tbody>
</table>

### D

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEEP</td>
<td>Dynamical Exascale Entry Platform</td>
</tr>
<tr>
<td>DEEP Architecture</td>
<td>Functional architecture of DEEP (e.g. concept of an integrated Cluster Booster Architecture)</td>
</tr>
<tr>
<td>DEEP Booster</td>
<td>Booster part of the DEEP System</td>
</tr>
<tr>
<td>DEEP Supercomputer</td>
<td>A future Exascale supercomputer based on the DEEP Architecture</td>
</tr>
<tr>
<td>DEEP System</td>
<td>The production machine based on the DEEP architecture developed and installed by the DEEP project</td>
</tr>
<tr>
<td>DGEMM</td>
<td>Double precision General Matrix Matrix multiplication</td>
</tr>
<tr>
<td>Dyninst</td>
<td>Tool developed by the University of Wisconsin that permits the insertion of code into a running program</td>
</tr>
</tbody>
</table>

### E

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC</td>
<td>European Commission</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>GFlop/s</td>
<td>Gigaflop, $10^{12}$ Floating point operations per second</td>
</tr>
<tr>
<td>Exascale</td>
<td>Computer systems or Applications, which are able to run with a performance between $10^{15}$ and $10^{18}$ floating point operations per second</td>
</tr>
<tr>
<td>EXTOLL</td>
<td>High speed interconnect technology for cluster computers developed by University of Heidelberg</td>
</tr>
<tr>
<td>Extrae</td>
<td>BSC’s instrumentation library for OmpSs and MPI applications</td>
</tr>
</tbody>
</table>

### F

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOP</td>
<td>Floating point Operation</td>
</tr>
<tr>
<td>FMA</td>
<td>Fused Multiply-Add</td>
</tr>
<tr>
<td>FWI</td>
<td>Full Waveform Inversion</td>
</tr>
</tbody>
</table>

### G

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
</tbody>
</table>

### H

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
</tbody>
</table>

### I

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>Intel GmbH Braunschweig, Germany</td>
</tr>
</tbody>
</table>

### K
Performance analysis tools ready

KNC: Knights Corner, Code name of a processor based on the MIC architecture
KNF: Knights Ferry, Intel first available processor based on the MIC

M
Mercurium: OmpSs’ source-to-source compiler
MIC: Intel Many Integrated Core architecture, also known as Xeon Phi
MPI: Message Passing Interface, API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages
MKL: Intel linear algebra library optimised for Intel architectures

N
Nanos++: OmpSs’ execution runtime.

O
OmpSs: BSC’s OpenMP SuperScalar programming model
OpenMP: Open Multi-Processing, application programming interface that supports multiplatform shared memory multiprocessing
OS: Operating System

P
Paraver: BSC’s visualisation tools
PM: Project Manager of the DEEP project
PMPI: MPI standard profiling interface
PMT: Project Management Team of the DEEP project
PR: Public Relations
Project Coordinator: Leading scientist coordinating and representing the DEEP project

T
ToW: Team of Work Package leaders within the DEEP project