HOlistic Performance System Analysis-EU

Deliverable D3.3

Lightweight Measurement Module

CONTRACT NO	HOPSA-EU 277463
INSTRUMENT	CP (Collaborative project)
CALL	FP7-ICT-2011-EU-Russia

Due date of deliverable:	August 15th, 2012
Actual submission date:	January 31st, 2013

Start date of project: 1 FEBRUARY 2011 Duration: 24 months

Name of lead contractor for this deliverable: GRS

Name of reviewers for this deliverable: Felix Wolf, Bernd Mohr

Abstract: This deliverable specifies the requirements and design of the lightweight measurement module. It specifies the internal working of the module, along with the output produced by it.

Revision 1.0

Project co-funded by the European Commission within the Seventh Framework Programme (FP7/2007-2013)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

Glossary

Abbreviation / acronym	**Description**
API	Application Programming Interface
HOPSA	HOlistic Performance System Analysis
HPC	High Performance Computing
I/O	Input/Output
LWM^2	Lightweight Measurement Module
MPI	Message Passing Interface (Programming Model for Distributed Memory Systems)
PMPI	Profiling MPI
PAPI	Performance Application Programming Interface (Library for accessing system hardware counters)
OpenMP	Open Multi-Processing (Programming Model for Shared Memory Systems)
POSIX	Portable Operating System Interface
CUDA	Compute Unified Device Architecture (Programming Model for nVidia Accelerators)
CUPTI	CUDA Profiling Tools Interface

1. Executive summary

This document describes the architecture of the Lightweight Measurement Module (LWM^2), including the profiling output of the tool, developed in the course of this project and defined by Task 3.3 of Work Package 3 of the EU FP7 project HOPSA. The HOPSA project (HOlistic Performance System Analysis) sets out for the first time to develop an integrated diagnostic infrastructure for combined application and system tuning. The documents provide an overview of the architecture of the lightweight measurement module and a snapshot of its internal working. This document then further explains the profiling output of LWM^2 and gives a glimpse of how to use the tool. First, the document lists the design requirements the LWM^2 has to fulfil in order to perform its role of an integrated profiler in the holistic environment of the project. It then describes the architecture of LWM^2, covering issues such as profiling methodology and storage of data. It then describes the concept of time slices, a novel method of data aggregation allowing cross application analysis. Finally, in architectural issues, the issue of thread handling and its effect on data storage and time slicing is discussed. At the end, the profiling output of LWM^2 is described and a glimpse of its usage is given.

2. Introduction

This document describes the architecture of the Lightweight Measurement Module (LWM^2), developed in the course of this project and defined by Task 3.3 of Work Package 3 of the EU FP7 project HOPSA. The document covers the technical design of LWM^2 in brief, describing the central issues of design requirements, profiling methodologies and data storage. Moreover, it also discusses issues like thread-safety and its effect on the architecture of the tool. At the end, it describes the profiling metrics collected by LWM^2.

2.1 The broader context: The HOPSA project

To maximise the scientific and commercial output of a high-performance computing system, different stakeholders pursue different strategies. While individual application developers are trying to shorten the time to solution by optimising their codes, system administrators are tuning the configuration of the overall system to increase its throughput. Yet, the complexity of today's machines with their strong interrelationship between application and system performance demands an integration of application and system programming.

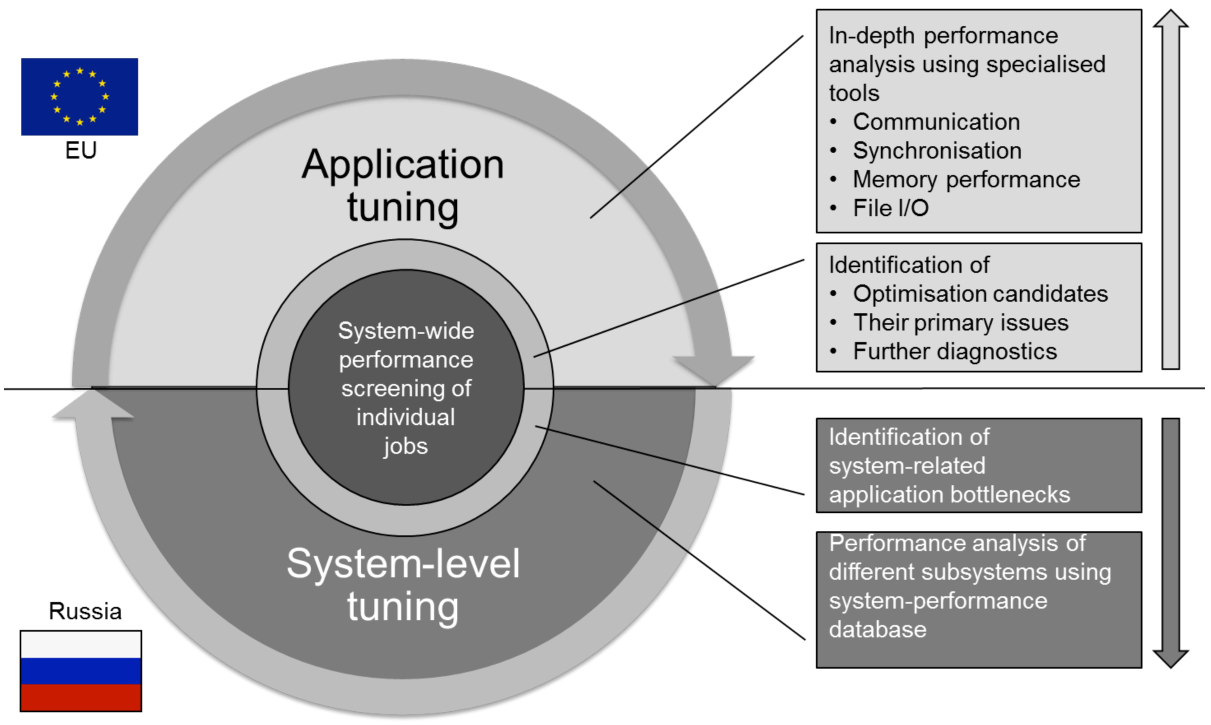
EU
Application tuning
System-wide performance screening of individual jobs
System-level tuning
Russia
In-depth performance analysis using specialised tools
• Communication
• Synchronisation
• Memory performance
• File I/O
Identification of
• Optimisation candidates
• Their primary issues
• Further diagnostics
Identification of system-related application bottlenecks
Performance analysis of different subsystems using system-performance database

Figure 1: System-level tuning (bottom), application-level tuning (top), and system-wide performance screening (centre) use common interfaces for exchanging performance properties.

The HOPSA project (HOlistic Performance System Analysis) therefore sets out for the first time to develop an integrated diagnostic infrastructure for combined application and system tuning. Using more powerful diagnostic tools application developers and system administrators will easily identify the root causes of their respective bottlenecks. With the HOPSA infrastructure, it is more effective to optimise codes running on HPC systems. More efficient codes mean either getting results faster or being able to get higher quality or more results in the same time.

The work in HOPSA is carried out by two coordinated projects funded by the EU under call FP7-ICT-2011-EU-Russia and the Russian Ministry of Education and Science, respectively. Its objective is the new innovative integration of application tuning with overall system diagnosis and tuning to maximise the scientific output of our HPC infrastructures. While the Russian consortium will focus on the system aspect, the EU consortium will focus on the application aspect.

At the interface between these two facets of our holistic approach, which is illustrated in Figure 1, is the system-wide performance screening of individual jobs, pointing at both inefficiencies of individual applications and system-related performance issues. The measurement module supposed to perform this task is the subject of this report.

2.2 About this document

This document provides a brief description of the architecture of the lightweight measurement module, developed during the course of the project. The document first lists the design requirements the LWM^2 has to fulfil in order to perform its role of a silent profiler in the holistic environment of the project. It then describes the architecture of LWM^2, covering issues such as profiling methodology and storage of data. It then describes the concept of time slices, a novel method of data aggregation allowing cross application analysis. Furthermore, the issue of thread handling and its effect on data storage and time slicing is discussed. At the end, the list of metrics collected by LWM^2 is described. This document can be considered as a high-level design document of LWM^2, which also lists major requirements of the tool.

3. Lightweight Measurement Module

The lightweight measurement module is a low overhead profiler developed during the course of the HOPSA project. It can profile applications without any modification by a user. This section first describes the role of LWM^2 in the holistic performance analysis environment. It then describes the architecture and the output generated by LWM^2.

3.1 Role of LWM^2 in holistic analysis environment

The Lightweight Measurement Module (LWM^2) functions as an integrated application profiler in the holistic performance analysis environment of the project. Its role is to mandatorily screen all the applications running on the system for performance. The information from application screening is stored in a central performance database, which can be accessed by a user to identify application performance problems and to select the appropriate tool for the application's performance problem. Figure 2 below presents the role of LWM^2 as a springboard for application performance analysis tools in the holistic environment. The information gathered from LWM^2 can also be used to identify inter-application interference.

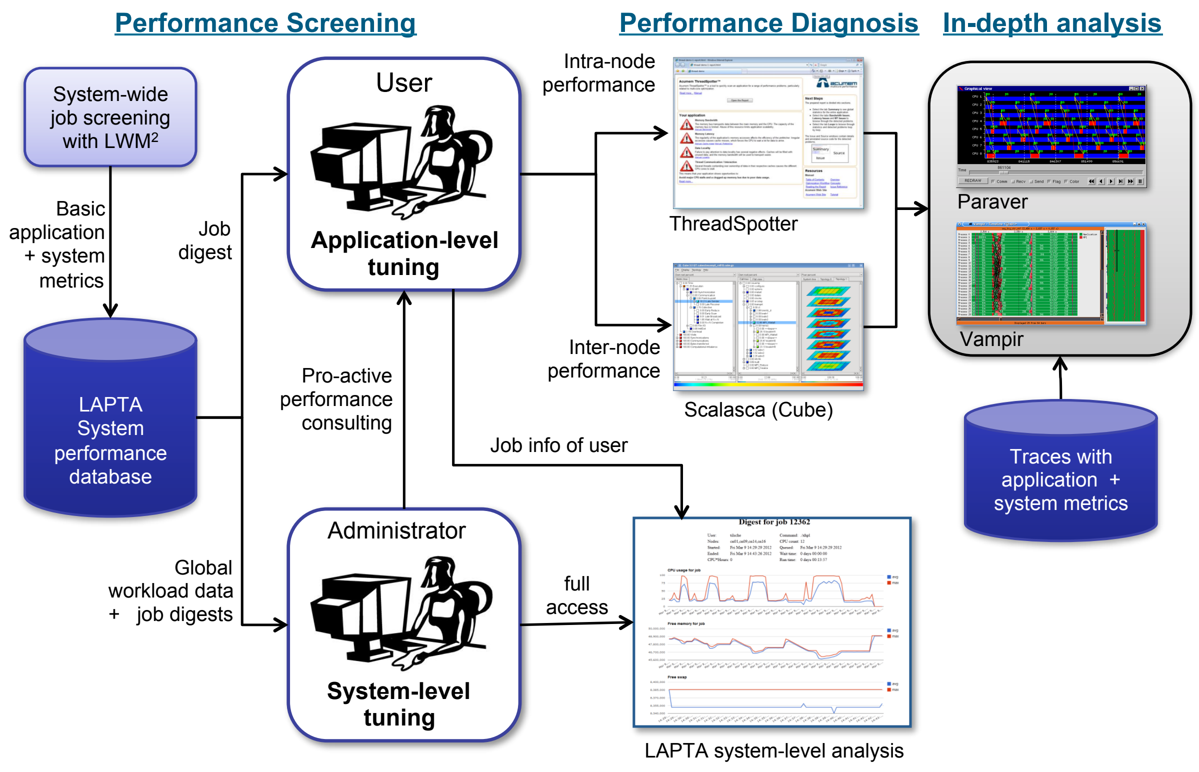

Figure 2: Overview of the planned performance analysis workflow.

The HOPSA performance analysis workflow is described in detail in deliverable D3.2.

3.1.1 Integration with LAPTA system

The lightweight measurement module forms the core of the automatic screening process on the holistic performance environment of the project. It integrates with the LAPTA system, developed by Moscow State University, to form a seamless analysis environment. At the end of execution of an application running in the holistic environment, LWM^2 generates multiple files containing the performance summary and the time-sliced information about the application's performance. The files are written to a specific directory, which is constantly scanned by an agent of the LAPTA system. The profiling data read by the agent is stored in the central database of the LAPTA system, which also contains performance data from hardware sources of the system.

3.1.2 Performance data access

The profiling information generated by LWM^2 can be accessed through a command line query or through a web interface. The web interface provides an online job digest, with graphical representation of metrics collected through LWM^2 and other performance data sources. The command line query uses HTTP POST request and executes a HOPLANG query to fetch the data, in a csv-list format. The following is a sample query, which fetches the MPI collective call count metric for a job with id 1868. The sample output is also given.

Query:

```
o = each x in post_lwm2_coll_call_count here x.time>0
  if x.task_id == '1868'
    yield x
  end
end
print o
```

Output:

```
n,node,task_id,time,value
0,cn05,1868,1357213230000000,10002
0,cn08,1868,1357213230000000,10002
0,cn05,1868,1357213230000000,10002
0,cn08,1868,1357213230000000,10002
0,cn05,1868,1357213230000000,10002
0,cn08,1868,1357213230000000,10002
```

3.2 Architecture

LWM^2 performs the role of an integrated application profiler in the holistic performance environment of the project. This role places some specific requirements on the design on LWM^2 that it has to fulfil.

3.2.1 Requirements

LWM^2 has to meet the following requirements

- Profiling of application without user interaction: As LWM^2 has to act as a silent profiler, it has to profile applications without requiring a user to modify the application. LWM^2 uses library preloading and dynamic function interposing to fulfil this requirement.

- Low resource utilization: As LWM^2 will be active by default, any resources consumed by LWM^2 will not be available to applications running on the system. For this reason, LWM^2 has to have a low overhead to keep the overall system utilization high.
- Basic performance information aggregation: The profiling information from LWM^2 will be stored in a central database, for all the applications running on the system. To keep storage requirements in realistic range, LWM^2 has to capture and provide basic, but useful, performance information about an application.

3.2.2 Profiling methodology

The lightweight measurement module uses a hybrid approach to profile an application. It samples the profiled application at regular intervals to keep track of application activity. To keep the overhead low, LWM^2 avoids stack unwinding at each application sample. Instead, it utilizes direct instrumentation to earmark regions of interest in an application. When an application is sampled, the earmarks are checked to identify the region of application execution. As a result, LWM^2 is able to profile application with reasonable knowledge of application activity while maintaining low overhead. This hybrid approach also allows LWM^2 to keep track of the time spent by an application in different regions of execution without directly measuring the time in these regions. All these approaches contribute to low overhead of LWM^2.

The hybrid profiling approach is also used to collect additional data of interest for some specific application activities. This includes the MPI communication calls and the amount of data transfer, the POSIX file I/O calls and associated data transfers, etc. This selected collection approach contributes to keeping the profiling information of LWM^2 small, as required by its role.

3.2.3 Supported technologies

The lightweight measurement module targets a typical HPC system, with a view on emerging technologies. It profiles MPI using the PMPI interface provided for profiling while it profiles POSIX file I/O by dynamic function interposing. It also measures the performance of multithreading in an application by estimating the effective thread count. Finally, CUDA applications are profiled through the CUPTI interface while the system hardware counters are profiled to collect the sequential performance information about the application.

MPI performance

The PMPI interface is used to directly instrument MPI calls. In direct instrumentation, only a few parameters of interests are recorded, while also setting earmarks for identifying MPI regions during sampling. The parameters of interest include communication information, like number of MPI collective and point-to-point communication calls, amount of data transferred in communication calls, etc.

File I/O performance

The file I/O parameters include both the MPI file I/O and POXIS file I/O parameters, which are instrumented separately. MPI file I/O instrumentation relies on PMPI, while POSIX file I/O is caught through dynamic symbol loading. Besides earmarking the region, the number of I/O operations and amount of data read/written are also recorded.

Multithreading performance

The multithreading performance of an application is estimated without making many assumptions about the underlying technology. A minimum assumption of a pthread-based runtime is made. The technique relies on the fact that every active thread is sampled separately, when an application is sampled. The ratio of how many samples were taken during an application execution to the maximum samples that can be taken of a single thread during an application’s execution gives a measure of effective threading performance of the application.

CUDA performance

The CUPTI interface is used to keep track of all the CUDA runtime calls. This provides a method to earmark CUDA activities on the host side. The runtime calls are also used to track memory transfers between the host and the device.

Sequential performance

The hardware counters provide an overview of the performance of an application on a single processor and on a single node. This sequential performance information is captured using the PAPI library for hardware-counter access.

3.2.4 Modular design

LWM^2 has a modular design, with each module responsible for profiling a different technology. The choice of hybrid profiling methodology means LWM^2 has both active and passive parts that gather profiling information. The passive parts are called as a result of direct instrumentation of application activities, while the active parts are those that are executed to sample the application. The profiling information from both of these methodologies is collected in a central storage module.

3.2.5 Time slices

The lightweight measurement module, when integrated into the holistic environment, also enables inter-application interference identification and correlation analysis. This is made possible through the novel concept of time slices. The profiling information collected by LWM^2 is aggregated and summarized for the whole execution of an application. To make cross application analysis possible, the profiling information is also aggregated for small segments of time, called time slices. This aggregation into time slices in effect creates small profiles of the application, as the application is executing, resulting in capturing the changing dynamics of the application. Figure 3 shows the aggregation of profiling information for the complete application execution and for time slices.

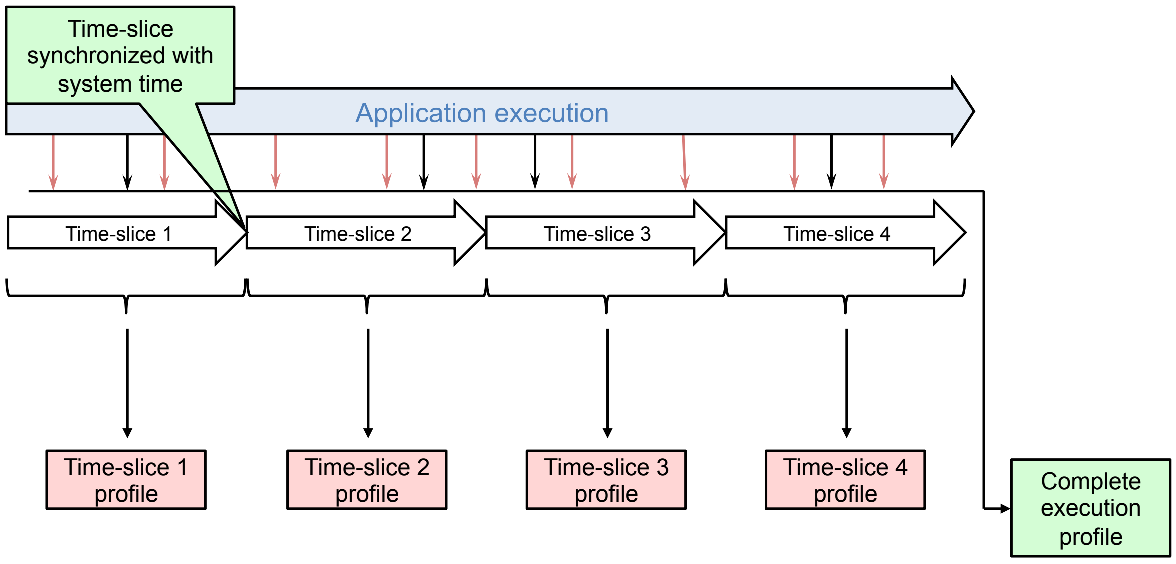

Figure 3: Aggregation of profiling information for the complete execution and for time slices.

The boundaries for aggregation of profiling information into time slices are synchronized along the global system time. This results in time-slice boundaries for all the applications executing on the system occurring at the same time. The small profiles created for the applications executing on the system are aligned to each other, and hence allow for analysis across applications.

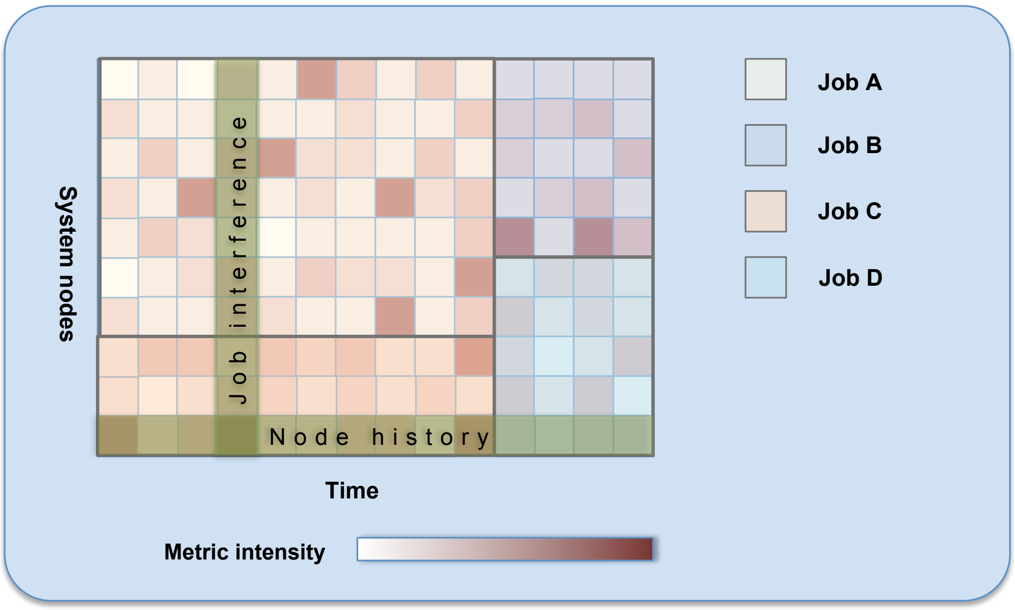

Figure 4: LWM^2 maps selected performance metrics collected during program execution onto a space-time grid. The space dimension consists of system nodes, while the time dimension consists of time slices, which are synchronized across the entire system to identify inter-application interferences from data of jobs executing simultaneously.

The time-sliced profiles of all the applications running at a time can be mapped on a space-time grid. Figure 4 shows such a grid, which makes inter-application interference evident. The discretised time axis constitutes the first dimension, the nodes of the system the second one. The purpose of organising the performance data in this way is threefold: First, by comparing the data of different jobs that were active during the same time slice, it becomes possible to see signs of interference between applications. Examples include reduced communication performance due to overall network saturation or low I/O bandwidth due to concurrent I/O requests from other jobs. Second, by looking at the performance data of the same node across a larger number of jobs and comparing it to the performance of other nodes during the same period, anomalies can be detected that would otherwise be hidden when analysing performance data only on a per-job basis. Third, collecting synchronised performance data from all the jobs running on a given system will open the way for new directions in the development of job scheduling algorithms that take the performance characteristics of individual jobs into account. For example, to avoid file-server contention and waiting time that may occur in its wake, it might be wiser not to co-schedule I/O-intensive applications. In this way, overall system utilisation may be further improved.

3.2.6 Thread handling

The lightweight measurement module is designed to profile multithreaded applications. As a result of the profiling methodology used, LWM^2 by itself has to be thread safe in its handling of profiling information to properly support multithreaded applications. The sampling approach and the direct instrumentation approach for profiling both offer different thread safety opportunities. In direct instrumentation, thread level mutual exclusion constructs can be used to safely store profiling information. Such features are not available in the sampling parts, where a special technique had to be developed to ensure thread-safety.

3.2.7 Architectural design

The LWM^2 is designed to be a thread-safe profiler, with both active and passive parts, storing profiling information in a central module, while also aggregating the information for small time segments. This led to some challenges in designing the software.

Thread-safe storage

The LWM^2 storage module stores all the profiling information, both for the complete application execution and for each time-slice. The complete application execution storage is fixed in size, and aggregates the information for the whole execution while the storage for time slices increases linearly with each time slice. The passive parts of LWM^2, utilizing the multithreading synchronization constructs available to them, store the captured profiling information directly into these two available storages.

To accommodate the active parts in a thread-safe design and to minimize cache line sharing among the threads, LWM^2 creates a separate storage for each thread of the application. This thread specific storage has parts both for storing complete application execution information and for time slices. Both the storages are fixed in size and are aggregated into the process wide storage module, at the end of execution for complete application storage and at the end of each time slice for time-slice storage.

Heartbeat thread

The time-sliced profile stored in thread specific storage is aggregated into the process wide storage at the end of each time slice. This is done through a heartbeat thread, which has access to multithreading synchronization constructs. The thread is activated at the end of each time slice and aggregates the time-sliced profiles of each thread into the process wide storage. A dual buffer approach is used to minimize the contention between the heartbeat thread and the passive parts of LWM^2 for the thread specific time-slice storage. One buffer is used to collect the profiling information, while the second buffer is accessed by the heartbeat thread to aggregate thread specific profiling information. The heartbeat thread switches these two buffers at the end of each time-slice before aggregating the performance information. Figure 5 shows the steps taken by the heartbeat thread at the end of a time-slice.

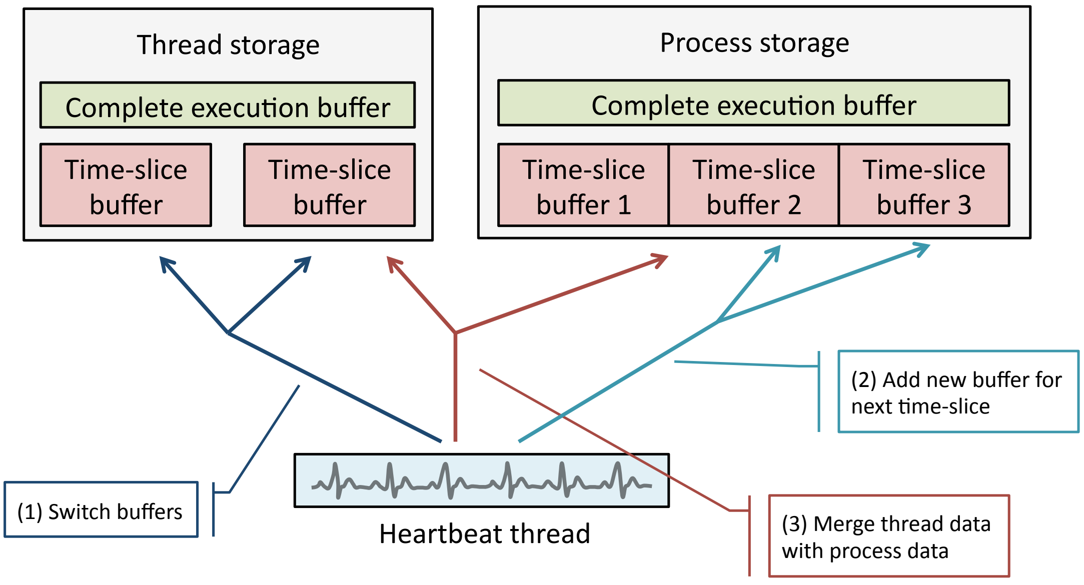

Figure 5: Storage handling at time-slice boundary.

Sampling data

The profiled application is periodically sampled by LWM^2 during execution. The information from the profiling is stored in the central storage module. In the case of multithreaded applications, each active thread is sampled separately. In order to avoid conflicts in the storage module, each thread stores its sampling information separately in its own thread specific storage. However, thread mutexes or locks are not available during application sampling, and hence the heartbeat thread at the end of a time-slice cannot safely aggregate the sampling information. As a result, the sampling information is only provided for the whole execution, and not for each time-slice. The information from the threads is aggregated into the process wide storage, at the end of an application execution, when all the threads have finished execution.

As a result of these techniques, a LWM^2 can safely handle the profiling information of multithreaded applications, while still maintaining a hybrid profiling approach

3.3 Usage & output

LWM^2 does not require any user effort to profile an application. It relies on library preloading for profiling. This requires only setting some environment variables, which can be configured by default in a batch system.

LWM^2 collects specific metrics while profiling an application. These are collected through a combination of sampling and direct instrumentation. These metrics are stored in a central database, and optionally displayed immediately after an application execution, in the form of a job digest. Below, a detailed list of metrics contained in the job digest is presented. For all metrics where it is applicable, the digest lists minimum, average and maximum values across processes. In addition to defining metrics, guidance in interpreting them and recommendations on further analyses is provided, if a given metric or group of metrics does not match expectations.

General information

- Duration of the job in terms of wall clock time
- Number of MPI processes

Message-passing performance

- Time spent in all MPI calls [%]
- Time spent in MPI point-to-point calls [%]
- Time spent in MPI collective calls [%]
- Average size of point-to-point messages [Byte]
- Average size of collective messages sent [Byte]
- Average size of collective messages received [Byte]
- Frequency of MPI point-to-point calls [/s]
- Frequency of MPI collective calls [/s]
- MPI point-to-point transfer rate [Byte/s]. Ratio of the number of bytes sent and the time spent in MPI point-to-point communication
- MPI collective transfer rate [Byte/s]. Ratio of the number of bytes sent and the time spent in MPI collective communication

In general, message passing means communication or synchronisation as opposed to computation and therefore does not directly contribute to the calculation of results. Therefore, communication should be minimized as much as possible and the fraction of time spent in MPI kept low. If the fraction of time spent in MPI calls grows with the number of processes, the application has usually a scalability problem. If communication is dominated by larger numbers of small messages, network latency may be the limiting factor. In contrast, if the majority of messages are large, the limiting factor may be network bandwidth. Asymmetries in the MPI time across processes, indicated by different minimum and maximum times, can be signs of load or communication imbalance, a performance property that usually prevents scaling to larger processor counts.

I/O performance

- Time spent in MPI file I/O calls [%]

- Time spent in POSIX file I/O calls [%]
- Amount of data written to files [Byte]
- Amount of data read from files [Byte]
- Write bandwidth [Byte/s]. Ratio of the number of bytes written to files and the time spent in write functions
- Read bandwidth [Bytes/s]. Ratio of the number of bytes written to files and the time spent in read functions

These metrics indicate whether the application places too much load on the I/O subsystem. The user should always check whether I/O of the given application coincides with I/O of other applications, which is visible in the Web-based digest. In such a case, the I/O performance may improve in subsequent runs when such interference is absent. In general, I/O performance is subject to variation and may change significantly between runs. This means, diagnosing an I/O bottleneck usually requires multiple runs under different overall load conditions.

Multithreaded performance

- Average number of threads for the execution: Ratio of the total number of samples and the number of samples taken on the master thread
- Total number of threads in the execution

The average number of threads tells whether the degree of concurrency is as expected. For example, long periods of sequential execution in OpenMP applications may degrade concurrency and limit the benefits of parallel regions for the overall program.

Sequential performance

- Average cycles per instruction (CPI)
- Fraction of floating-point operations among all instructions [%]
- Level1 data cache hit ratio
- Last-level miss frequency

Sequential-performance metrics tell how well the cores of the underlying machine are utilized. If the cycles per instructions are much higher than the theoretical minimum, then memory access latency or pipeline hazards may be the reason. Also, some operations such as complex floating-point operations may simply take longer than others. The fraction of floating-point operations tells to which degree floating-point performance is the dominant theme. A low Level1 hit ratio usually indicates low locality and may explain a high CPI value. The last-level miss frequency is equivalent to the frequency of main-memory accesses and may point to memory-bandwidth saturation. Note that a platform may miss some of the hardware counters required for the full set of sequential performance metrics or that some of the required hardware counters cannot be measured simultaneously. In this case, LWM^2 provides only a subset of the above metrics.

CUDA performance

- Time spent in CUDA calls [%]
- Average data volume transferred from host to device [Byte]
- Average data volume transferred from device to host [Byte]
- Frequency of data transfers [/s]

These metrics provide just a very rough indicator of CUDA performance.

4. Conclusions

The HOPSA project creates an integrated diagnostic infrastructure for combined application and system tuning. At the centre of this infrastructure is the lightweight measurement module, which acts as a silent profiler. It mandatorily screens all the applications running on the system, providing a feedback on application performance and guidance in selection of specialized performance analysis tools. This document describes the architecture of LWM^2, the requirements it had to fulfil to fill in the role of a silent profiler and the strategies used by the tool to achieve thread safety. The final LWM^2 software package will be available for download at the project website (www.hopsa-project.eu) and will be distributed as part of the HOPSA UNITE package (deliverable D3.4).

Beyond the lifetime of the project, the HOPSA infrastructure is supposed to collect large amounts of valuable data on the performance of individual applications as well as the system workload as a whole. It will be of interest in three ways: to tune individual applications, to tune the system for a given workload, and finally to observe the evolution of this workload over time. The latter will allow the effectiveness of our strategy to be studied. An open research issue to be tackled on the way will be the reliable tracking of individual applications, which may change over time, across jobs based on the collected data. In this way, it will become possible to document the performance history of code projects and demonstrate the effects of our tool environment over time.