Continuous System-wide Profiling of High Performance Computing Applications

VISHWANATH DUGANI
Continuous system-wide profiling of High Performance Computing applications

Abstract

Profiling of an application identifies parts of the code being executed using the hardware performance counters thus providing the application’s performance. Profiling has long been standard in the development process focused on a single execution of a single program. As computing systems have evolved, understanding the bigger picture across multiple machines has become increasingly important. As supercomputing grows in pervasiveness and scale, understanding parallel applications performance and utilization characteristics is critically important, because even minor performance improvements translate into large cost savings. The study surveys various tools for the application. After which, Perfminer was integrated in SCANIA’s Linux clusters to profile CFD and FEA applications exploiting the batch queue system features for continuous system wide profiling, which provides performance insights for high performance applications, with negligible overhead. Perfminer provides stable, accurate profiles and a cluster-scale tool for performance analysis. Perfminer effectively highlights the micro-architectural bottlenecks.
Continuous system-wide profiling of High Performance Computing applications

Abstract

Continuous system-wide profiling of High Performance Computing applications

Contents

1 Introduction 9
  1.1 Background ................................................. 9
  1.2 Motivation ................................................. 9
  1.3 Objectives ................................................. 10
  1.4 Contributions ............................................. 10
  1.5 Organization .............................................. 10
  1.6 Methodology .............................................. 11
    1.6.1 Strategy and Research Design ......................... 11
    1.6.2 Data Collection and Analysis Methods ................. 11

2 Literature review 11
  2.1 Tools .................................................. 12
  2.2 Selection ............................................... 15

3 Implementation 15
  3.1 Profiling ............................................... 15
  3.2 Applications ........................................... 16
  3.3 Batch system Integration .................................. 16
    3.3.1 CPU Frequencies .................................... 17
    3.3.2 Job Environment .................................... 17
    3.3.3 SYSCTL ........................................... 17
    3.3.4 Pre-execution Script ................................ 18
    3.3.5 Post Execution ..................................... 18
  3.4 Process ................................................. 20
  3.5 Database .............................................. 22
  3.6 Visualization ........................................... 22

4 Results and Analysis 23
  4.1 HPC Infrastructure ...................................... 23
  4.2 CPU Utilization ......................................... 23
  4.3 Resources ............................................... 24
  4.4 Memory Access .......................................... 27
    4.4.1 Stores ........................................... 27
    4.4.2 Loads ............................................ 28
    4.4.3 Cache Performance .................................. 29
  4.5 Instruction composition ................................ 30
  4.6 Branch Mis-predictions ................................ 31
  4.7 CPU or I/O bound ....................................... 32
  4.8 User time .............................................. 33
  4.9 Overhead ............................................... 33

5 Conclusions and Future Work 35
  5.1 Ethics, Reliability, Validity, Generalization and Limitations ... 35
  5.2 Future work ............................................ 36

A Appendix 41

B Appendix 42
List of Tables


List of Figures

1. Process of Profiling .......................................................... 21
2. Job execution time (seconds) and time (seconds) spent in contesting for a resource. .................................................... 24
3. Percentage time (seconds) spent in contesting for a resource. .......................................................... 25
4. Percentage of the job time lost due to Instruction Queue was full. ................................................... 26
5. Percentage of the job time lost due to Store Buffers were full .................................................... 27
6. Memory Stores .................................................................. 28
7. Memory Loads .................................................................. 29
8. Cache Performance .......................................................... 30
9. Type of instructions .......................................................... 31
10. Branch mis-predictions ..................................................... 32
11. The application is CPU bound or I/O bound ...................... 33
12. Time usage as per users ..................................................... 33
13. Overhead introduced by the profiler .................................. 34
Acknowledgment

This thesis is accomplished with kind help of a lot of people to whom I wish express my sincere thanks. It has been a period of learning and growing for me, not only in the this study, but also on a personal level. This thesis has had a lasting impact on me. I would like to reflect on the people who have supported and helped me so much throughout this period.

First, I wish express my gratitude to HPC team at SCANIA, for allowing me to be a part of this study. Special thanks to Xavier Aguliar for guiding me throughout the journey and Erwin for his insights of the report. And a special shoutout to Phil Mucci and Tushar Mohan for providing the tool, source code and guidance throughout the integration stage. Finally, I would like to thank my parents for supporting me throughout my studies at KTH and being there for me.
1 Introduction

1.1 Background

Continuous system-wide profiling aims to extract information about the applications executing on Linux clusters. The profiler leverages the hardware performance counters of the CPU to enable profiling of a wide variety of interesting statistics such as cache misses, memory accesses details, resource stalls etc. The profiler counts all the micro-architectural hardware events for the application, executing on a specific hardware.

Profiling has long been standard in the development process. Most profiling tools focus on a single execution of a single program. As computing systems have evolved, understanding the bigger picture across multiple machines has become increasingly important. As super computing grows in pervasiveness and scale, understanding parallel applications performance and utilization characteristics is critically important, because even minor performance improvements translate into large cost savings [3].

Performance analysis tools are designed to give system administrators an overview of utilization of existing hardware. This is intended to address inefficient use of compute resources and better distribution of resources to the users. It also helps to improve the application development process by creating a feedback system between system administrators and developers. Finally, it should also help in the maintenance of the HPC infrastructure as a whole and support decisions for future investment.

1.2 Motivation

A significant portion of Scania’s yearly IT budget is spent on software licensing costs for HPC applications. The licenses for these applications are not granted via a per-user, per-core or per-run policy, but rather dependent on the complexity of the calculations that they perform. As such, the yearly expenditures for these licenses are based entirely on their performance on Scania’s infrastructure.

The motivation for the study is a suite of ISV (Independent Software Vendor) applications used in the development and simulation of technologies and solutions in the structural, thermodynamic and chemical combustion domains (i.e., Abaqus). There is no accountability in the way of measuring their performance in a consistent, quantitative and automatic manner. The current practice of "paper and pencil" tracking of wall-clock time of selected jobs is both infrequent and time-consuming. Furthermore, this method itself is of limited use since the vendor can rather simply dismiss the results via the claim of a "sub-optimal HW/SW infrastructure" as the process does not provide any data to refute this claim. As such, Scania has as greatly impaired ability to hold the ISV’s accountable for the performance of their codes. The ISVs have a strong financial incentive.

The second is a set of changing computational resources consisting of multiple platforms each with differing versions of hardware and software technology. (i.e. Intel/Linux/GPU clusters). Scania, like many similar organizations, strives to stay current (if not ahead) of the high-performance technical computing curve through relatively frequent (1 system every 12-18 months) purchases of large computer systems to run the above codes. These systems (consistent with com-
mon industry practice) are retired every 3-5 years, as the cost of maintaining them versus their performance versus newer hardware justifies their specifications and replacement.

1.3 Objectives

The first is, surveying and choosing an existing tool that can be relevant to the project, and analyzing if it can be adapted to the characteristics and requirements of SCANIA’s system. Choosing from a number of tools (both closed and open source). Hence, a study of the SCANIA’s cluster environment, understanding the user interface, job handling, batch queuing system and compilers is crucial in choosing the right tool. Choosing the right tool entails understanding the intricacies of integrating the tools with little to no modification to the existing setup at SCANIA and with very low overhead.

The second, Integration of the tool in SCANIA environment in a way that it profiles every job non-intrusively. One of the important requirements for the study is that the tool is invisible or non-intrusive to the user. Hence, studying different possible ways to instrument the application in the cluster will be a vital step of the whole project. In this step we will ensure that every application being run on the cluster is continuously profiled, with no exceptions.

And lastly, setting up database and visualization tools to get an overview of the data collected. The files generated during and after the process of profiling need to be handled effectively to extract useful information. Hence, a database that can handle thousands of lines of data is imperative. Secondly, a visualization tool should be able query the database depending on the user’s need. The visualization tool is the front-end or the top layer of every performance analysis tool as it allows users to get a bird’s eye view of the cluster’s performance.

1.4 Contributions

This study is focused towards contributing specifically to SCANIA’s environment. The first contribution is: This study has encouraged SCANIA to effectively formulate the motivation and requirement for a performance analysis tool. The second contribution is: This study has laid down the ground work for the future in understanding the batch system to non-intrusively integrate the performance analysis with the least overhead(0.8%) without making much changes to the existing system. The third contribution is: The study analyzes performance by understanding the behavior of hardware’s(Intel) micro architecture for HPC applications. Event-based analysis that helps identify the most significant hardware issues affecting the performance of an application. Consider this analysis type as a starting point when you do hardware-level analysis.

1.5 Organization

In section 1 the paper gives some background, the motivation which will explain the purpose of the study, the objectives and contribution it made at the end of the study. In section 2 the methodology of the study will be discussed in detail. The philosophy behind the study. The approach taken to solve the problem. The strategy, research design, data collection and analysis methods. The ethics, reliability, validity and limitations of the study. In section 3 the
Continuous system-wide profiling of High Performance Computing applications

literature regarding related work and existing tools relevant to the problem at hand are discussed in detail. The decision of choosing a tool is also addressed in this section. The profilers that were considered to be deployed at SCANIA are also briefly explained. In section 3 the deployment of the selected profiler is discussed in steps and in detail. In section 4 we discuss results obtained from the data-set, that the profiler was able to extract from the PMU units. In section 5, the conclusion and future prospects of the work are discussed.

1.6 Methodology

1.6.1 Strategy and Research Design
Quantitative methods are mainly used in the data collection process of research. The strategy to collect data was using the hardware performance counters or the PMU provided by the Intel IA-32 architecture built in the hardware. The performance of the whole cluster will be determined by weighing some specific events that provide insights about the capacity of the cluster and application’s behavior throughout the execution.

1.6.2 Data Collection and Analysis Methods
The data collected for this study was limited to one specific application (ABAQUS) running on the cluster. Due to time constraints and other limitations, the results in this study will focus on one application and one set of hardware in the cluster. The quantitative data collected during the course of this study, whilst still in its raw form, it is useless and convey little information to most people. Therefore, results will be analyzed using graphs and would be drawn out to analyze all data.

2 Literature review

There are two basic categories of performance analysis tools profilers and tracers. Profilers provide a summary of execution statistics and/or events. Profilers give an overview of the overall performance of the program, often broken down to the functions, loops or even user-specified sections. They often use periodic sampling during a run with little impact on overhead, so longer runs to improve the accuracy of the results are encouraged. They are best at exposing bottlenecks and hot-spots in overall execution. Profiling are not fit for time-dependent behavior, but its measurement and analysis scale easily for long executions. For this reason, tools employ profiling.

Tracers, on the other hand, often do provide this profile information along with breaking it down along a temporal context. They record a much larger stream of information during a run in order to reconstruct the dynamic behavior over a finely resolved time-sequence. Often with parallel runs, a separate log file is created for each process or thread being traced. These files can quickly become huge [43], and as such, tracing is not recommended for lengthy runs or at large scales. However, the detail they provide allows a realm of performance tuning options not available through profiling alone. The large amounts of data that tracers generate require a visualization tool or other type of interpreter that presents it comprehensively to users in order to make sense of it [4]. several tools
use tracing to measure how an execution unfolds over time. Tracing can provide valuable insight into phase and time-dependent behavior and is often used to detect MPI communication inefficiencies [7, 8, 9, 35, 38, 45, 42].

Most tools [26, 28, 30, 25, 41] also support both profiling and tracing. Because either profiling or tracing may be the best form of measurement for a given situation, tools that support both forms have a practical advantage. A true performance analysis tool should indicate where and which optimization will yield the greatest cross-platform and machine-specific benefits, including actual code modification suggestions.

Performance Analysis: These tools typically provide more than simple profiling and tracing capabilities. They can include functionality such as multiple methods of data visualization, the calculation of derived metrics, integration of performance data with a database, network performance modeling, etc. [5, 10, 12]. Performance tools measure the same dimensions of an execution, they will vary in measurement methodology. TAU [39], OPARI [29], and Pablo [33] instrument the code during build process. Model-dependent methods use instrumentation of libraries [17, 23, 34, 37, 43]. Some tools analyze unmodified application binaries by dynamically instrumenting the jobs [19, 20, 28, 32] or library pre-loading [16, 24, 30, 36, 41]. These different measurement approaches affect a tool’s ease of use, but more importantly affect its potential for accurate and scalable measurements. Many scalable performance tools manage data by collecting summaries based on synchronous monitoring (or sampling) of library calls (e.g., [43, 44]) or by profiling based on asynchronous events (e.g., [14, 16, 31]).

Tools for measuring parallel application performance are typically model dependent, such as libraries for monitoring MPI communication (e.g., [43, 44, 40]), interfaces for monitoring OpenMP programs (e.g., [17, 29]). Perfminer’s call path profiler uniquely combines pre-loading (to monitor unmodified dynamically linked binaries), asynchronous sampling (to control overhead), and binary analysis (to assist handling of unruly object code) for measurement.

2.1 Tools

In this section we will briefly look the tools surveyed for the study. The tools surveyed are as follows:
Continuous system-wide profiling of
High Performance Computing applications

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Profiler</th>
<th>Tracer</th>
<th>Perf analysis tool</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gprof</td>
<td></td>
<td></td>
<td></td>
<td>Standard unix/linux profiling utility.</td>
</tr>
<tr>
<td>Intel Advisor</td>
<td></td>
<td></td>
<td></td>
<td>Performance analysis tool for threaded codes (no-MPI).</td>
</tr>
<tr>
<td>mpiP</td>
<td></td>
<td></td>
<td></td>
<td>Lightweight MPI profiling tool.</td>
</tr>
<tr>
<td>memP</td>
<td></td>
<td></td>
<td></td>
<td>Lightweight memory profiling tool.</td>
</tr>
<tr>
<td>HPCToolkit</td>
<td></td>
<td></td>
<td></td>
<td>Integrated suite of tools for parallel program performance analysis.</td>
</tr>
<tr>
<td>Open—Speedshop</td>
<td></td>
<td></td>
<td></td>
<td>Full featured parallel program performance analysis tool set.</td>
</tr>
<tr>
<td>TAU</td>
<td></td>
<td></td>
<td></td>
<td>Full featured parallel program performance analyses toolkit.</td>
</tr>
<tr>
<td>Intel Vtune</td>
<td></td>
<td></td>
<td></td>
<td>Full featured parallel performance analysis tool.</td>
</tr>
<tr>
<td>Intel Profiler</td>
<td></td>
<td></td>
<td></td>
<td>Compiler based loop and function performance profiler.</td>
</tr>
<tr>
<td>PAPI</td>
<td></td>
<td></td>
<td></td>
<td>A standardized and portable API for accessing performance counter hardware.</td>
</tr>
<tr>
<td>Papiex</td>
<td></td>
<td></td>
<td></td>
<td>A PAPI-based performance profiler</td>
</tr>
<tr>
<td>Perfminer</td>
<td></td>
<td></td>
<td></td>
<td>Performance analysis tool for threaded codes (no-MPI).</td>
</tr>
</tbody>
</table>
Continuous system-wide profiling of High Performance Computing applications

**HPCToolkit**  HPCToolkit is an integrated suite of tools for measurement and analysis of program performance on computers ranging from multicore desktop systems to the largest supercomputers. The tool uses statistical sampling of timers and hardware performance counters, HPCToolkit collects measurements of a program’s work, resource consumption, and inefficiency and attributes them to the full calling context in which they occur. HPCToolkit works with multilingual, fully optimized applications that are statically or dynamically linked. Since HPCToolkit uses sampling, measurement has low overhead (1-5 percent) and scales to large parallel systems. HPCToolkit’s presentation tools enable rapid analysis of a program’s execution costs, inefficiency, and scaling characteristics both within and across nodes of a parallel system. HPCToolkit supports measurement and analysis of serial codes, threaded codes (e.g. pthreads, OpenMP), MPI, and hybrid (MPI+threads) parallel codes [13].

**Intel VTune Amplifier**  Intel VTune Amplifier is a performance analysis tool for users developing serial and multithreaded applications. VTune Amplifier helps you analyze the algorithm choices and identify where and how your application can benefit from available hardware resources [28].

**TAU**  TAU (Tuning and Analysis Utilities) is capable of gathering performance information through instrumentation of functions, methods, basic blocks, and statements as well as event-based sampling. All C++ language features are supported including templates and namespaces. The API also provides selection of profiling groups for organizing and controlling instrumentation. The instrumentation can be inserted in the source code using an automatic instrumentor tool based on the Program Database Toolkit (PDT), dynamically using DyninstAPI, at runtime in the Java Virtual Machine, or manually using the instrumentation API [39].

**Integrated Performance Monitoring (IPM)**  IPM is a portable profiling tool for parallel codes. It provides a low-overhead performance profile of the performance aspects and resource utilization in a parallel program. Communication, computation, and IO are the primary focus. While the design scope targets production computing in HPC centers, IPM has found use in application development, performance debugging and parallel computing education. The level of detail is selectable at run-time and presented through a variety of text and web reports [5].

**PerfMiner**  Minimal Metrics (http://www.minimalmetrics.com) is dedicated to the performance optimization of profit-critical processes and technology. PerfMiner seamlessly collects performance data and its presentation via a web browser. Perfminer uses data collector(database), mmpercollect(event based sampling) and papiex in its back-end. The primary objectives of the PerfMiner project are:

- Allow the HPC scientist to improve application performance
- Provide actionable insights for HPC administrators to improve cluster utilization
Continuous system-wide profiling of
High Performance Computing applications

- Performance regression analysis capability

The system comprises of the following core components:

- Data collector, such as mmperecollect or papiex
- Data store
- A RESTful front-end

2.2 Selection

The selection of the tool was based on two key factors, support and requirements of SCANIA. The first requirement: The tool should not compromise the existing setup, in other words it, should be integrated with little to no changes in the existing operations. The second requirement is the overhead of the performance analysis tool. Since SCANIA’s production systems run 24x7 throughout the year, it is critical that the tool does not add an overhead of more than 1%. The third requirement for the tool is to be able to handle both statically and dynamically linked binaries. After considering the requirements, the potential candidates for the study were HPCtoolkit, IPM and Perfminer. Narrowing down the list to one of the tools went through a few iterations. In each iteration one tool from the shortlist is tried.

In the first iteration IPM was tried. IPM is a standalone profiler with no database and with an old visualization framework. SCANIA’s unique setup demanded considerable changes to its configuration file. The tool also needed considerable work in building a new database and visualization tool. After taking into account the amount of effort and time constraints of the project, IPM had to be eliminated from the shortlist. In the second iteration, HPCtoolkit was tried. The HPCtoolkit was easy to integrate, but configuring it with new hardware event-list from Intel was quite a challenge. Furthermore, SCANIA’s proprietary applications were packaged with their own version of MPI compilers which were built with symbols that could not be identified by HPCtoolkit. In the second iteration it was clear that the performance analysis tool for SCANIA needs a custom modification to work around the proprietary applications. Before heading into the last iteration, HPCtoolkit and Minimal Metrics were contacted and asked for support. Perfminer’s makers came forward and offered to guide through the integration process of the tool. Additionally, Perfminer had the potential to satisfy SCANIA’s requirements. A unanimous decision from all the stakeholders of the study was to go ahead with Perfminer.

3 Implementation

3.1 Profiling

Profiling illuminates how the existing invocation of an algorithm executes. It allows a software developer to improve the performance of that invocation. It does not offer much insight about how to change an algorithm, as that really requires a better understanding of the problem being solved rather than the performance of the existing solution. In this step Perfminer’s mmperecollect is used to extract the hardware events as the application executes.
3.2 Applications

SCANIA is a truck manufacturer, and therefore their software is dominated and encompassed by two broad categories: structural analysis and fluid dynamics analysis. Fluid dynamics analysis includes applications used to perform computational fluid dynamics (CFD), while structural analysis encompasses applications for analyzing structures, including explicit and implicit finite element analysis (FEA).

FEA (Finite Element Analysis) and CFD (Computational Fluid Dynamics) are both branches of CAE (computer-aided engineering) whereby we’re using the power of computers to solve what can be the most complicated engineering problems.

FEA and CFD involves some of the highest levels of mathematics, engineering, computer programming and computer software and hardware specifications.

The manufacturing segment is one of the largest markets for high performance computing, globally. In fact, the large product manufacturing sub-segment is the biggest vertical in commercial HPC. All leading automotive, aerospace, and heavy equipment manufacturers have employed HPC for decades, using the technology to design and test their products.

Using digital simulations allows manufacturers to reduce costs by replacing costly development of physical models with virtual ones during various stages of the product development workflow. Potential benefits include improved product quality, shorter time to market, and reduced manufacturing costs.

In the context of SCANIA, one application (Abaqus) dominates the cluster. Users run this application interactively across SCANIA systems. A custom made script handles the job for the sake of ease. It takes in various inputs such as the job script, type of analysis, number of nodes, and other input files. The application uses its own compilers, the reason being it helps them save time on resolving unwanted symbols and linked libraries. The application is designed such that inputs are funneled to a binary in the job script. The binary compiles the user inputs and dlopen() mpi library. Since the Abaqus binaries provided to SCANIA do not directly link to the mpi libraries, profiling them requires major source code modifications in the profiler.

3.3 Batch system Integration

This section explains how the profiler was integrated with the batch queuing system Load sharing facility (LSF).

The integration was accomplished using the LSF’s job starter feature. A job starter is a specified wrapper script or executable program that typically performs environment setup for the job, and then it calls the job itself, which inherits the execution environment created by the job starter. LSF controls the job starter process, rather than the job. One typical use of a job starter is to customize LSF for use with specific application environments.

At SCANIA, the job starter is placed in the directory "/opt/lsf/prod/utils/starter" for the production environment.

The following are some of the major steps involved in setting the system to profile any executing application in the cluster.
Continuous system-wide profiling of
High Performance Computing applications

3.3.1 CPU Frequencies
The CPU frequencies for all the nodes allocated by LSF are recorded.
The script for recording the frequencies is provided in the appendix.

```sh
#!/bin/sh

cd /sys/devices/system/cpu

for cpu in 'ls -d cpu[0-9]*'; do
  for file in $cpu/cpufreq/*; do
    if [ -f $file -a -r $file ]; then
      echo "$file:'cat $file'
    fi
done
done
```

3.3.2 Job Environment
In this step we extract information about the job such as the job ID, the name of the script, or the host name for instance. The snippet of the script below performs this information extraction.

**Job ID**

```sh
if 'PERFMINDER_JOB_ID' in os.environ:
  return os.environ['PERFMINDER_JOB_ID']
return False
```

**Job Script name**

```sh
if 'PERFMINDER_JOB_SCRIPT_NAME' in os.environ:
  return os.environ['PERFMINDER_JOB_SCRIPT_NAME']
return False
```

**Host Name**

```sh
host = "";
try:
  host = check_output('hostname', shell=True).rstrip().replace(',',PM_REPLACE_COMMA)
except CalledProcessError as e:
  print >> sys.stderr, "Error determining hostname: ", e
return host
```

3.3.3 SYSCTL
In this step we ensure some of the kernel level parameters are set in order to extract performance information from the system.

```
/sbin/sysctl -a
```
Continuous system-wide profiling of High Performance Computing applications

kernel.perf event paranoid = 2: you can’t take any measurements. The perf utility might still be useful to analyse existing records with perf ls, perf report, perf timechart or perf trace.
kernel.perf event paranoid = 1: you can trace a command with perf stat or perf record, and get kernel profiling data.
kernel.perf event paranoid = 0: you can trace a command with perf stat or perf record, and get CPU event data.
kernel.perf event paranoid = -1: you get raw access to kernel tracepoints (specifically, you can map the file created by perf event open, I don’t know what the implications are).

CPU model check  In this step we check if the hardware being used is supported by the profiler. In the snippet of script below we see that ”txt” files have the events supported by Intel for the Intel Nehalem micro architecture. If any events are missing from the text file, the events will not be registered.

```python
if model == "6-45":
    event_list.extend(EVENTLIST_PERF)
    event_list.extend(read_event_list('EVENTLIST_INTEL.txt'))
    event_list.extend(read_event_list('EVENTLIST_INTEL_NHM.txt'))
```

At this point we have set the stage for profiling the job.

3.3.4 Pre-execution Script

A part of the job starter script that executes before the job is called the pre-execution. In this section of the script job specific and LSF environment variables are read and stored for references.

Start time stamp  This time stamp is at the beginning of the starter script.

```bash
export PERFMINER_BEGIN_TIMESTAMP=$(date -u +%s%6N)
```

Output Directory

```bash
export PERFMINER_OUTPUT_DIR=${PERFMINER_BATCH_NAME}-${PERFMINER_JOB_ID}.perfminer
    debug "export PERFMINER_OUTPUT_DIR=${PERFMINER_OUTPUT_DIR}"
    debug "mkdir ${PERFMINER_OUTPUT_DIR}"
    mkdir ${PERFMINER_OUTPUT_DIR}
```

Start time stamp

```bash
export PERFMINER_BEGIN_TIMESTAMP=$(date -u +%s%6N)
```

The pre-execution script setup the stage for profiling and dumping the extracted data to a specific directory.

3.3.5 Post Execution

In the post execution script the steps followed were as follows.
Continuous system-wide profiling of High Performance Computing applications

Stop time stamp  At the beginning of the post-execution script the job execution time is recorded.

Count files that match  The function counts the files generated by the profiler that are relevant.

```bash
prefix=$1
suffix=$2
shopt -s nullglob
pattern
jobfiles=(${prefix}.[0-9][0-9]*$suffix)
echo ${#jobfiles[@]}
shopt -u nullglob
```

Check files that match  This part of the script ensure that only the relevant files are handled.

```bash
total=$1
prefix=$2
suffix=$3
n=0
found=0
while [ $n -le $total ]; do
    name=${prefix}.${n}${suffix}
    if [ -s ${name} ]; then
        found=$((found+1))
    fi
    n=$((n+1))
done
echo $found
```

get num ranks  The function counts the number of ranks.

```bash
start=$(date +%s)
while ! ls $PM_NODE_WAIT_FILE > /dev/null 2>&1 ; do
togo=$(check_timeout $start $PM_NODE_WAIT_TIMEOUT "while waiting for $PM_NODE_WAIT_FILE")
if [ $togo == "0" ]; then
    warn "Could not determine number of ranks in job"
    return 1
else
    debug "sleeping 1s waiting for $PM_NODE_WAIT_FILE, $togo seconds to go before timing out"
fi
sleep 1
done
cat $PM_NODE_WAIT_FILE
```

Collate files  The profiler generates lots of files. Collation of files into a single file, helps the data to be organized and easy to be uploaded to the database.
Continuous system-wide profiling of
High Performance Computing applications

if [ $ran_papiex -eq 0 ]; then
    debug "waiting for mmpercollect to write the perf files for collation"
    mmpc_prefix="${PERFMINER_JOB_BINARY}.mmpc"
    (cd "$perfminer_outdir"; if node_wait $mmpc_prefix ""; then mm-perf-process-csv.py "$PERFMINER_JOB_BINARY"; fi)
else
    debug "this is a papiex run; no collation of perf files needed"
fi
(cd "$perfminer_outdir"; file_name="$(ls | grep mmpc.0)");
bin_name="${file_name%.mmpc.*}"; mm-perf-process-csv.py $verbose "$bin_name"
(cd "$perfminer_outdir"; if node_wait node_env .csv; then collate_csv.py $verbose *node_env.*.csv > collated_node_env.csv; fi)
(cd "$perfminer_outdir"; if node_wait sysctl .csv; then collate_csv.py $verbose *sysctl.*.csv > collated_sysctl.csv; fi)
(cd "$perfminer_outdir"; if node_wait cpufreq .csv; then collate_csv.py $verbose *cpufreq.*.csv > collated_cpufreq.csv; fi)

**clean up**  This part of the script ensures that all the unwanted files are deleted after collation.

if [ $keep_files -eq 0 ]; then
    debug "removing per-rank files"
    (cd "$perfminer_outdir"; rm -f node_env.* sysctl.* cpufreq.*)
else
    debug "not removing per-rank files as -d or -k set"
fi
ddebug "removing lock file: $PM_NODE_WAIT_FILE"
rm -f "$perfminer_outdir/$PM_NODE_WAIT_FILE"

**Stop time stamp**  At the beginning of the post-execution script time is recorded save the starter-script execution time.

### 3.4 Process

Intel processors already provide the capability to monitor performance events. In order to obtain a more precise picture of CPU resource utilization we rely on the dynamic data obtained from the so-called performance monitoring units (PMU) implemented in Intel's processors.

Figure 1 shows the profiling process from the start to the end of a job. We can see in the figure that the user’s job passes through various LSF daemons for scheduling and resource allocations. After the daemons are done the job
Continuous system-wide profiling of High Performance Computing applications

profiling begins in parallel.

User
The user submits the job to the cluster with all the necessary inputs for the job

bsub < <job file>

LSF Daemons
As soon as the user submits the job to the cluster the job is passed through various LSF daemons. The daemons can be configured in the lsf.conf file. The lsf.conf file also helps to create test queues. In order for the job to be profiled the lsf.conf file need to the point to the directory of the job starter script.

The mbatchd is the master batch daemon and is responsible for job requests and dispatch. The Master Batch Daemon runs on the master host. It is responsible for the overall state of jobs in the system and the state of the jobs can be queried using the command “bjobs”. It receives job submission, and information query requests. It manages jobs held in queues. It also dispatches jobs to hosts as determined by mbschd.

The mbschd is the master scheduler daemon is responsible for job scheduling and it runs on the master host. It works with mbatchd and is started by mbatchd.

The sbatchd is the job execution Slave Batch Daemon running on each server host. It receives the request to run the job from mbatchd and manages local
execution of the job. It is responsible for enforcing local policies and maintaining the state of jobs in the host. The sbatchd forks a child sbatchd for every job. The child sbatchd runs an instance of res to create the execution environment in which the job runs. The child sbatchd exits when the job is complete.

Res is the job execution Remote Execution Server (res) running on each server host. Accepts remote execution requests to provide transparent and secure remote execution of jobs and tasks. Users around SCANIA’s different facilities interact through this server.

3.5 Database

The database was packaged with the profiler written in mongoDB. The profiled data is imported to the database in the post-execution script. The database spits out JSON files when queried.

3.6 Visualization

The visualization feature was readily available with perfminer. It was built with the D3.js JavaScript library. The visualization tool is hosted on a HTTP server for the users.
4 Results and Analysis

In this section we will discuss the results and findings of the study. The results shown below were derived from SCANIA’s production environment, and are an overview of the performance for 7 days in a cluster that runs the Abaqus application. The jobs were run on the same type of hardware (Intel Ivy bridge 6-45). All nodes and cores were clocked at 2.6GHz. The results were derived following the Performance Analysis Guide offered by Intel.

Abaqus applications can differ from one another in three key categories, that is the kind of FEA/CFD analysis, number of cores used and execution time. Hence, the only way to find a common ground or a metric for the results (hardware events) is in-terms of percentage. All the results are normalized per core. Also, most results provide median in percentage to find the central tendency since the data-set is very skewed and also because median is not affected by outliers, hence highlighting the outliers. The results also provide deviation in percentage for the data-set to understand the fluctuations in the corresponding hardware event. All the key hardware events relevant to this study are listed in the appendix.

4.1 HPC Infrastructure

At SCANIA the Hardware and software infrastructure was as follows:

Cluster’s hardware The cluster’s hardware architectures are classified as follows:
- PRODUCTION: Intel Ivy Bridge
- TEST: Intel Ivy Bridge
- ALEPH: Intel Nehalem

Operating system Linux redhat (Santiago) Enterprise Server release 6.4

Kernel The linux kernel version 2.6.32

File System SCANIA uses NFS. The profiler and extracted data were placed in the shared file system.

Batch Queuing system Load Sharing Facility- LSF stands for LOAD SHARING FACILITY. LSF manages, monitors, and analyzes the workload for a heterogeneous network of computers and it unites a group of computers into a single system to make better use of the resources on a network. Hosts from various vendors can be integrated into a seamless system.

LSF is based on clusters. A cluster is a group of hosts. The clusters are configured in such a way that LSF uses some of the hosts in the cluster as batch server hosts and some others as client hosts.

4.2 CPU Utilization

\[
\text{Utilization} = \frac{\text{CPUsUsed}}{\text{CPUsAllocated}} \times 100
\]  \hspace{1cm} (1)
Continuous system-wide profiling of  
High Performance Computing applications

The CPU utilization for all the recorded jobs is 100%. No events with utilization less than 100% were recorded.

4.3 Resources

In this section we see the performance bottle-necked by the limitation of resources available for the application to execute. However, contention for resources will always exist as applications have become more complex, but it is crucial to reduce these contentions to the least. In the hardware’s micro architecture there are two main kinds of stalls.

Retired Stalls: This metric is defined as a ratio of the number of cycles when no micro-operations are retired to all cycles. In the absence of performance issues, long latency operations, and dependency chains, retire stalls are insignificant. Otherwise, retire stalls result in a performance penalty.

Execution Stalls: Execution stalls may signify that a machine is running at full capacity, with no computation resources wasted. Sometimes, however, long-latency operations can serialize while waiting for critical computation resources.

The CYCLE_ACTIVITY.CYCLES_NO_EXECUTE hardware event counts the cycles for total execution stalls and the time wasted due to these execution stalls is plotted in the graph below. Figure 2 you see the contains the job IDs in the X-axis and the wall-clock time for total execution stalls in the Y-axis.

![Figure 2: Job execution time(seconds) and time (seconds) spent in contesting for a resource.](image)

In the graph below (Figure 3) we see what percentage of the total execution time of the job was limited by resources in the micro-architecture of the hardware. The X-axis is the job IDs and the Y axis is the percentage of the total execution time. The median for this data set is 20.1% with a deviation of 5.03%. Job number 4, 6 and 36 were bottle-necked due to resources and hence have room for optimization in mathematical operations [15]. For example, consider replacing 'mul' operations with left-shifts, or try to reduce the latency of memory accesses.
ILDSTALL.IQFULL hardware event gives the Stall cycles because Instruction Queue(IQ) is full. In the graph below (Figure 4) the X-axis contains the job IDs and the Y-axis represents the percentage of total stalls due to the IQ is full. L1 instruction stall cycles: In a shared-memory machine, large code working set, branch mis-prediction, including one caused by excessive use of virtual functions, can induce misses into L1I and so cause instruction starvation that badly influence application performance. Job number 4, 6, and 44 are bottlenecked by IQ. The dataset had a median of 6.7%.

Figure 3: Percentage time (seconds) spent in contesting for a resource.
Continuous system-wide profiling of
High Performance Computing applications

Figure 4: Percentage of the job time lost due to Instruction Queue was full.

RESOURCESTALLS.SB hardware event counts the cycles stalled due to no store buffers (SB) available. In the graph below (Figure 5) the X-axis contains the job IDs and the Y-axis represents the percentage of total stalls due to Store Buffer (SB) is full. The root causes could be excessive cache misses or false sharing. Job number 4, 6 and 44 are bottlenecked by SB as well. The dataset has a median of 6.3%.
4.4 Memory Access

Memory access is costly in terms of time for any application, optimizing the application to reduce memory access always improves performance, hence we will look at the instruction break up for loads and stores in this section. Memory access (Loads and stores) in L1 to main memory can cost on average up to 27 nano seconds in penalty. Memory access shows a fraction of cycles spent waiting due to demand load or store instructions.

4.4.1 Stores

Job 4 and 6 have the most store operations than any other job. In the graph below (Figure 6), job IDs are in the X-axis and the percentage of store instructions are in the Y-axis.

The deviation is 2.08% and the Median is 6.85% for this data set.
Continuous system-wide profiling of High Performance Computing applications

4.4.2 Loads

In the graph below (Figure 7) the X-axis is job IDs and the Y-axis represents the percentage of load instructions. Deviation is 8.28% and Median is 24.2%.
4.4.3 Cache Performance

In this section we see how the applications behaved in terms of cache. The cache architecture for Ivy bridge is L1 Data cache has a size 32 KB, 64 B/line, 8-WAY. L1 Instruction cache has a size of 32 KB, 8-WAY, 64 B/line. L2 cache has a size of 256 KB, 64 B/line, 8-WAY. Last Level cache or L3 cache has a size of 8 MB, 64 B/line. In the Ivy bridge architecture, the potential miss penalty is at least 16 cycles.

The MEM_LOAD_UOPS RETIRED.L1_HIT hardware event registers the retired load uops with L1 cache hits as data sources. MEM_LOAD_UOPS RETIRED.L2_HIT registers retired load uops with L2 cache hits as data sources. MEM_LOAD_UOPS RETIRED.L2_HIT registers retired load uops with L3 cache hits as data sources. MEM_LOAD_UOPS RETIRED.HIT.LFB registers retired load uops which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.

In Figure 8 we can clearly see a poor performance by job number 47. It is confirmed false sharing, where it keeps missing on L1 and fetches the data from L2. This was a test application called abaqus-frequency, which is used only on a couple of occasions for test purposes. In figure 8, the X axis represents job Ids and the Y axis represents the cache hit percentage.

False sharing: Most high performance processors, insert a cache buffer between slow memory and the high speed registers of the CPU. Accessing a memory location causes a slice of actual memory (a cache line) containing the memory location requested to be copied into the cache. Subsequent references to the same memory location or those around it can probably be satisfied out of the cache until the system determines it is necessary to maintain the coherency.
between cache and memory. However, simultaneous updates of individual elements in the same cache line coming from different processors invalidates entire cache lines, even though these updates are logically independent of each other. Each update of an individual element of a cache line marks the line as invalid. Other processors accessing a different element in the same line see the line marked as invalid. They are forced to fetch a more recent copy of the line from memory or elsewhere, even though the element accessed has not been modified. This is because cache coherency is maintained on a cache-line basis, and not for individual elements. As a result, there will be an increase in interconnect traffic and overhead. Also, while the cache-line update is in progress, access to the elements in the line is inhibited. This situation is called false sharing. If this occurs frequently, performance and scalability of a parallel application will suffer significantly.

![Cache Performance](image)

**Figure 8: Cache Performance.**

### 4.5 Instruction composition

In this section we see the breakup of all the types of instructions the application is made up of. The avx\_insts.all hardware event registers the number of vector instructions in the application. BR\_INST\_RETIRED.ALL\_BRANCHES registers all the branch instructions in the application. fp\_assist.x87\_input registers the number of X87 assists due to input value.

Figure 9 shows job IDs in the X axis and the percentage of breakp in the Y-axis.
Continuous system-wide profiling of
High Performance Computing applications

Figure 9: Type of instructions

4.6 Branch Mis-predictions

Intel’s branch mispredict hardware event BR_MISP RETIRED.ALL_BRANCHES counts all mis-predicted macro branch instructions retired. In the Ivy Bridge architecture one misprediction on average costs 14 cycles [15]. When a branch mispredicts, some instructions from the mispredicted path still move through the pipeline. All work performed on these instructions is wasted since they would not have been executed had the branch been correctly predicted. This metric represents slots fraction the CPU has wasted due to branch misprediction. These slots are either wasted by uOps fetched from an incorrectly speculated program path, or stalls when the out-of-order part of the machine needs to recover its state from a speculative path. A significant proportion of branches are mispredicted, leading to excessive wasted work or Backend stalls due to the machine need to recover its state from a speculative path. Developers should consider ways to make the algorithm more predictable or to reduce the number of branches. If the developer can move ‘if’ statements as high as possible in the code flow (that is, as early as possible, and containing as much as possible). If using ‘switch’ or ‘case’ statements, put the most commonly executed cases first. Avoid using virtual function pointers for heavily executed calls [15].

In the figure below (figure 10) X axis represents the job IDs and Y axis represents the percentage of mis-predictions of all branch instructions. The median for this dataset is 0.58% and the deviation for this data set is 0.18%. We see that job 47 has the worst branch mis-predictions and hence can be further optimized with the suggestions mentioned above.
4.7 CPU or I/O bound

In this section we address if the application is I/O bound or CPU bound. \texttt{cycles:u} registers the number of cycles the application spent on the CPU. CPU Bound means the rate at which process progresses is limited by the speed of the CPU. \texttt{cycles:k} registers the number of cycles the application spent in the kernel for I/Os. I/O Bound means the rate at which a process’s progress is limited by the speed of the I/O subsystem.

In the graph below (figure 11) X axis represents job Ids and Y axis represents percentage of cycles. In the graph we see that job number 47 is more I/O bound than the rest of the jobs.
Continuous system-wide profiling of High Performance Computing applications

4.8 User time

Our performance framework provides also statistics about the cluster usage per user. Figure 12 shows such user of the cluster per user. X axis is the user-name and Y axis is time used in seconds.

4.9 Overhead

Overhead is an important parameter when it comes to performance monitoring. It is desirable not to perturb the application too much with the measurement process. In this study the maximum overhead was found to be 0.8%. Since the PMU unit is embedded with the hardware, the low overhead is justifiable.

**Overhead Time** Over head is the time spent for the sake of profiling. In the context of SCANIA, the overhead can be defined as follows:
Continuous system-wide profiling of High Performance Computing applications

JobStarterExecutionTime is the time spent executing the job starter script (refer 3.3).

\[ \text{TotalExecutionTime} = \text{JobStarterExecutionTime} \]  \hfill (2)

\[ \text{OverHead} = \text{JobStarterExecutionTime} - \text{JobExecutionTime} \]  \hfill (3)

\[ \text{PercentageOverhead} = \frac{\text{JobExecutionTime}}{\text{JobStarterExecutionTime}} \times 100 \]  \hfill (4)

In the figure (Figure 13) below X axis represents job IDs and Y axis represents overhead percentage.

Figure 13: Overhead introduced by the profiler
5 Conclusions and Future Work

This dissertation studies the performance of HPC applications on the Intel’s Ivy bridge architecture. Perfminer profiles high performance applications by counting the events and exposing various aspects of such applications. Perfminer was able to point out the poorly performing applications. The profiler gathered information about utilization, memory access, resources, memory, cache, and instruction composition. These parameters tell us how the application behaved and what were the bottlenecks. Hence, helping developers to tune the performance for more efficiency.

However, putting performance in a perspective that could be understood by an individual with no knowledge on computer architecture is yet to be addressed. The future work does address the need to formulate metrics that are more simplistic in nature. These metrics were not simplified considering the time frame of this project. As it had become clear early in the project that the tool had to be custom modified for SCANIA’s use most of the effort was invested in developing and integrating the profiler. On the other hand, Perfminer was effectively integrated in the cluster non-intrusively with the help of LSF. Perfminer also stayed within the limits of the overhead requirements of SCANIA, which was less than 1%.

5.1 Ethics, Reliability, Validity, Generalization and Limitations

Ethics One ethical issue has been identified and raised with respect to this study. The company is secretive about some aspects of the HPC applications, such as the specific models being used in FEA and CFD analysis, which SCANIA may not like its competitors to know about. The information about the FEA and CFD applications are not revealed in this study, hence maintaining the ethics.

Reliability The performance of the study is based on the data of PMU events. The study assumes that the data from the performance counters is accurate. The data obtained are experimental values while expected may be different from Intel based on theory or simulations. Data obtained in this study follow a particular test method in SCANIA. It is possible that different values can be obtained under other conditions. Some latencies are difficult or impossible to measure accurately, especially for memory access and type conversions that cannot be chained.

Statistical sampling does not provide 100% accurate data. When the tool collects an event, it attributes not only that event but the entire sampling interval prior to it (often 10,000 to 2,000,000 events) to the current code context. For a big number of samples, this sampling error does not have a serious impact on the accuracy of performance analysis and the final statistical picture is still valid. But if something happened for very little time, then very few samples will exist for it. This may yield seemingly impossible results, such as two million instructions retiring in 0 cycles for a rarely-seen driver [15].

Validity The study is validated by bench-marking simple applications whose performance is already known. For example, the results of a simple application
Continuous system-wide profiling of
High Performance Computing applications

for calculating the value of pi were compared with expected results and results obtained.

**Generalization** The study cannot be generalized for performance on other linux clusters as the set conditions at SCANIA are very unique.

**Limitations** The study was not able to extract data on parallel performance (mpi cycles), and change the source code of the profiler to extract data from all applications, as the applications were handled non-conventionally. In this context non-conventionally means, the applications were packaged with compilers from the ISVs to eliminate the step of linking libraries and additionally these applications dynamically load other binaries during run-time. The cores at SCANIA are connected to various ports for memory, bus, etc. The study was not able to accurately map the ports for section (4.9). No datasheets or documentation were found to eliminate this limitation.

Also, due to the time constraints, results across different hardware were not tested. The profiler is integrated in the batch system (LSF) layer at SCANIA. The profiler extracts data on various hardware events as the application is executing. However, the tool is not yet ready to be used by IT professionals as the metrics are at a very lower level of abstraction.

5.2 Future work

In this section we will discuss the future prospects of this study at SCANIA and try to understand where the study should be heading in the future. The first step is to discuss and formulate metrics that are more appropriate for IT professionals. In the results the discussed CPU metrics may not be appropriate for all audiences. More simplistic metrics that can cater to a non-technical audience is imperative.

Adding statistical models to the front-end for better analysis. User’s must be able to statistically subject a chosen data set, which might highlight some of the underlying behavioral patterns of some applications. E.g.: The average, median and standard deviation of cache misses for the month of January. The statistical versatility of the front-end without doubts will leverage faster business decisions.

Dynamic job handling, for more efficient job handling. Since all the applications at SCANIA are proprietary, all ISVs have their own way of handling the jobs. The performance analysis tool must be able to adapt to all handling methodologies. In such a scenario providing the profiler with the right input becomes challenging. Hence designing the profiler that adapt to handle new applications with new behavior will be a crucial step in the future.

The learning of application’s behavior will effectively assist scheduling and redistribution of hardware resources. There are different kinds of application and different hardware specifications at SCANIA, by knowing the behavior applications can be redistributed to the hardware it performs the best. Example: what applications on the cluster are I/O bound? What applications execute vector instructions? Can we run this application on GPUs for optimal performance?

Correlating performance metrics with monetary benefits. In the production environment at SCANIA time is critical. Hence, correlating poor performance to
time and time with monetary penalties continuously will help the IT professional
with business decisions and penalizing ISVs for poor code.

References

[1] Philip J. Mucci, Daniel Ahlin, Johan Danielsson, Per Ekman, and Lars Ma-
linowski PerfMiner: Cluster-Wide Collection, Storage and Presentation of
Application Level Hardware Performance Data.


GOOGLE-WIDE PROFILING:A CONTINUOUS PROFILING INFRA-
STRUCTURE FOR DATA CENTERS.

and Technologies.


[6] David Skinner Introspection with IPM. High Performance Computing and
Communications and 2013 IEEE International Conference on Embedded and
Ubiquitous Computing (HPCC, EUC), 2013 IEEE 10th International Con-
ference on. IEEE, 2013.

[7] Aguilar, Xavier, Karl Fürlinger, and Erwin Laure. MPI trace compres-
sion using event flow graphs. European Conference on Parallel Processing.

[8] Aguilar, Xavier, Karl Fürlinger, and Erwin Laure. "Automatic on-line de-
tection of MPI application structure with event flow graphs.” European Con-

[9] Aguilar, Xavier, Karl Fürlinger, and Erwin Laure. "Online MPI Trace Com-
pression Using Event Flow Graphs and Wavelets.” Procedia Computer Sci-

[10] Aguilar, Xavier, Karl Fürlinger, and Erwin Laure. "Visual MPI perfor-
ance analysis using event flow graphs.” Procedia Computer Science 51


[12] Erich Birngruber, Petar Forai, Aaron Zauner Total Recall: Holistic Metrics
for Broad Systems Performance and User Experience Visibility in a Data-
Intensive Computing Environment

Crummey, N. R. Tallent HPCTOOLKIT : Tools for performance analysis of
optimized parallel programs

1990.
Continuous system-wide profiling of High Performance Computing applications


Continuous system-wide profiling of High Performance Computing applications


Continuous system-wide profiling of High Performance Computing applications


The following script was used to plot the data for all performance measurement.

```python
import plotly.plotly as py
from plotly.graph_objs import *
py.sign_in('username', 'api_key')
trace1 = Scatter(
    x=[],
    y=[],
    name='Percentage time wasted(Resource)',
    uid='530f07',
    xsrc='vishwanath.dugani:95:aa1a3f',
    ysrc='vishwanath.dugani:95:93270a')
trace2 = Scatter(
    x=[],
    y=[],
    name='Percentage time wasted(Cache)',
    uid='c63901',
    xsrc='vishwanath.dugani:95:aa1a3f',
    ysrc='vishwanath.dugani:95:172f15')
trace3 = Scatter(
    x=[],
    y=[],
    name='Median for resource stalls',
    uid='abe145',
    xsrc='vishwanath.dugani:95:aa1a3f',
    ysrc='vishwanath.dugani:95:4371eb')
trace4 = Scatter(
    x=[],
    y=[],
    name='Median for cache miss',
    uid='66530a',
    xsrc='vishwanath.dugani:95:aa1a3f',
    ysrc='vishwanath.dugani:95:36eda1')
data = Data([trace1, trace2, trace3, trace4])
layout = Layout(
    autosize=True,
    height=496,
    title='Percentage wasted',
    width=1121,
    xaxis=XAxis(
        autorange=True,
        range=[0, 81],
        title='Job ID',
    )
)
```

Continuous system-wide profiling of High Performance Computing applications

```python
    type='linear',
    yaxis=YAxis(  
      autorange=True,
      range=[-2.613656209222223, 49.92237009522223],
      title='Percentage',
      type='linear'
    )
  }
fig = Figure(data=data, layout=layout)
plot_url = py.plot(fig)
```

## Appendix

The following is an example of the abaqus script.

```bash
#!/bin/bash

#JOBNAME
#BSUB -J balk

. /share/apps/env/clusters/euler/abaqus_stageing.sh
JOBID=$LSB_JOBID
ln -s /net/$HOSTNAME/scratch/$USER/abaqus_job_$JOBID balk\_$JOBID

#BSUB -n 8

# New serial cluster
##BSUB -R "select[n_opti_aba]"
##BSUB -m "aleph-c-2-1"
##BSUB -m "ts208-c-4-2 ts208-c-4-3"

#Queue name
#BSUB -q vish

# APPLICATION PROFILE
#BSUB -app abaqus

# Sends email when the job is dispatched
#BSUB -u vishwanath.dugani@scania.com
echo "$PERFMIMER_BASE_DIR"
echo ""
unset DISPLAY

bpost -i 11 -d "VERSION=6142" $LSB_JOBID
bpost -i 12 -d "JOBNAME=balk" $LSB_JOBID
GetFromDir=$STARTDIR
INCCLUDE_FILENAME=.INCLUDE.$LSB_JOBID
echo "-- $INCCLUDE_FILENAME" > $GetFromDir/$INCCLUDE_FILENAME
echo "+ balk.inp" >> $GetFromDir/$INCCLUDE_FILENAME
```
Continuous system-wide profiling of High Performance Computing applications

sort $GetFromDir/$INCLUDE_FILENAME | uniq > uniqfile.$JOBID

if [[ $RET_CODE -ne 0 ]]; then
  echo "Rsync failed with exit status[$RET_CODE]"
  exit $RET_CODE
fi

if [ $? -ne 0 ]; then
  echo "stage up failed, aborting"
  exit 1
fi

if [ -f ".abaqus_v6.env" ]; then
  rm .abaqus_v6.env
fi

cd $SCRATCHDIR
if [ -f ".abaqus_v6.env" ]; then
  mv .abaqus_v6.env abaqus_v6.env
fi

declare -a absolute_path_array=( ${absolute_path_array[@]} )
declare -a absolute_filepath_array=( ${absolute_path_array[@]} )
argument1=$(echo ${absolute_path_array[@]})
argument2=$(echo ${absolute_path_array[@]})
abaqus_inputfile_modification balk.inp "$argument1" "$argument2"
machinememory=$(free -g|grep Mem|awk '{print $2}')
machinememory=$(echo $machinememory*0.8|bc|cut -d. -f1)
memory_req_byuser=75
if [ -f abaqus_v6.env ]; then
  COM=
  if [[ $COM = mp_rsh_command ]]; then
    sed "/$COM/d" abaqus_v6.env >> abaqus_v6.env_2
cp abaqus_v6.env_2 abaqus_v6.env
  rm abaqus_v6.env_2
  fi
  echo "memory="$machinememory "gb"" >> abaqus_v6.env
else
  touch abaqus_v6.env
  chmod 775 abaqus_v6.env
  echo "memory="$machinememory "gb"" >> abaqus_v6.env
fi

export TYPEOFANALYSIS=Non-linear
if [[ Non-linear = Linear ]]; then
Continuous system-wide profiling of
High Performance Computing applications

abaqus_linear balk.inp

SHARE/apps/abaqus/Linux/Commands/abq6142 job=balk input=balk.inp
interactive gpus=1 cpus=$LSB_DJOB_NUMPROC
ExitStatus=$?

# Time logs
# Read system time, total time and wallclock time from .dat file
if [-f balk.dat ]; then
DATE=20160404_14:40:25
SYSTEM_TIME=$(tail balk.dat | grep SYSTEM | awk '{ print $5 }')
TOTAL_CPU_TIME=$(tail balk.dat | grep TOTAL | awk '{ print $6 }')
WALLCLOCK_TIME=$(tail balk.dat | grep WALLCLOCK | awk '{ print $5 }')
typeofanalysis_log=Non-linear
version_log=6142
user_log=sssmjf
echo "${user_log} $LSB_JOBID ${LSB_QUEUE} ${version_log}
$LSB_DJOB_NUMPROC
$SYSTEM_TIME $TOTAL_CPU_TIME $WALLCLOCK_TIME ${DATE}
${typeofanalysis_log}" >>
/grp/techsim/chassicalc/abaqus_cluster_job_logs/abaqus_logs
fi

# INCLUDES
GetToDir=$STARTDIR
EXCLUDE_FILENAME=.EXCLUDE.$LSB_JOBID
INPUTFILECHANGE=INPUTFILECHANGE.$LSB_JOBID

stage_down $GetToDir
RET_CODE=$?
if [[ $RET_CODE -eq 12 || $RET_CODE -eq 23 ]]; then
  echo "File copying is fail. Please check your disk space.
   It may be full."
  . /share/apps/env/clusters/euler/par_abaqus_stageing.sh
RUNDIR2=/cluster/seasonas13/sssmjf/stageback_$LSB_JOBID
GetDir=$RUNDIR2
ln -sf /cluster/seasonas13/sssmjf/stageback_$LSB_JOBID stageback_$LSB_JOBID
EXCLUDE_FILENAME=.EXCLUDE.$LSB_JOBID
INPUTFILECHANGE=INPUTFILECHANGE.$LSB_JOBID

echo "- $EXCLUDE_FILENAME" > $GetToDir/$EXCLUDE_FILENAME
echo "- homedir" > $GetToDir/$EXCLUDE_FILENAME
stage_back2 $GetToDir

44/51
Continuous system-wide profiling of High Performance Computing applications

cd $STARTDIR
ln -sf $RUNDIR2 stageback.$LSB_JOBID
if [[ ${SCRATCHDIR} != "" ]]; then
  rm -rf $SCRATCHDIR/
  rm $STARTDIR/balk\_$JOBID > /dev/null 2>&1
fi
elif [[ $RET_CODE -eq 0 ]]; then
  if [[ ${SCRATCHDIR} != "" ]]; then
    rm -rf $SCRATCHDIR/
    rm $STARTDIR/balk\_$JOBID > /dev/null 2>&1
  fi
else
  echo "Rsync is failed with exit status [$RET_CODE] Sending more information in mail"
fi

echo "$LSB_JOBID $RET_CODE" >> /grp/techsim/chassicalc/abaqus_cluster_job_logs/abaqus_rsync_logs
cd $STARTDIR
rm $INCLUDE_FILENAME
rm $EXCLUDE_FILENAME
exit $ExitStatus

C Appendix

The following are the hardware-event references from INTEL.

RESOURCESTALLS.ANY
Resource-related stall cycles

RESOURCESTALLS.ROB
Cycles stalled due to re-order buffer full.

RESOURCESTALLS.RS
Cycles stalled due to no eligible RS entry available.

RESOURCESTALLS.SB
Cycles stalled due to no store buffers available. (not including draining form sync).

FP_COMP_OPS_EXE.SSE_PACKED_DOUBLE
Number of SSE* or AVX-128 FP Computational packed double-precision uops issued this cycle

FP_COMP_OPS_EXE.SSE_PACKED_SINGLE
Number of SSE* or AVX-128 FP Computational packed single-precision uops issued this cycle
Continuous system-wide profiling of
High Performance Computing applications

FP_COMP_OPS_EXE.SSE_SCALAR_DOUBLE
Number of SSE* or AVX-128 FP Computational scalar
double-precision uops issued this cycle

FP_COMP_OPS_EXE.SSE_SCALAR_SINGLE
Number of SSE* or AVX-128 FP Computational scalar
single-precision uops issued this cycle

FP_COMP_OPS_EXE.X87
Number of FP Computational Uops Executed this cycle. The number
of FADD, FSUB, FCOM, FMULs, integer MULs and IMULs, FDIVs, FPREMs,
FSQRTS, integerDIVs, and IDIVs. This event does not distinguish
an FADD used in the middle of a transcendental flow from a s

ICACHE.HIT
Number of Instruction Cache, Streaming Buffer and Victim Cache Reads.
both cacheable and noncacheable, including UC fetches

ICACHE.IFETCH_STALL
Cycles where a code-fetch stalled due to L1 instruction-cache
miss or an iTLB miss

ICACHE.MISSES
Instruction cache, streaming buffer and victim cache misses

OTHER_ASSISTS.AVX_STORE
Number of GSSE memory assist for stores. GSSE microcode assist
is being invoked whenever the hardware is unable to properly handle
GSSE-256b operations.

OTHER_ASSISTS.AVX_TO_SSE
Number of transitions from AVX-256 to legacy SSE when
penalty applicable.

OTHER_ASSISTS.SSE_TO_AVX
Number of transitions from SSE to AVX-256 when penalty applicable.

ARITH.FPU_DIV
Divide operations executed

ARITH.FPU_DIV_ACTIVE
Cycles when divider is busy executing divide operations

BACLEARS.ANY
Counts the total number when the front end is resteered, mainly
when the BPU cannot provide a correct prediction and this is corrected
by other branch handling mechanisms at the front end.

BR_INST_EXEC.ALL_BRANCHES
Continuous system-wide profiling of High Performance Computing applications

Speculative and retired branches

**BR_INST_EXEC.ALL_CONDITIONAL**
Speculative and retired macro-conditional branches

**BR_INST_EXEC.ALL_DIRECT_JMP**
Speculative and retired macro-unconditional branches excluding calls and indirects

**BR_INST_EXEC.ALL_DIRECT_NEAR_CALL**
Speculative and retired direct near calls

**BR_INST_EXEC.ALL_INDIRECT_JUMP_NON_CALL_RET**
Speculative and retired indirect branches excluding calls and returns

**BR_INST_EXEC.ALL_INDIRECT_NEAR_RETURN**
Speculative and retired indirect return branches.

**BR_INST_EXEC.NONTAKEN_CONDITIONAL**
Not taken macro-conditional branches

**BR_INST_EXEC.TAKEN_CONDITIONAL**
Taken speculative and retired macro-conditional branches

**BR_INST_EXEC.TAKEN_DIRECT_JUMP**
Taken speculative and retired macro-conditional branch instructions excluding calls and indirects

**BR_INST_EXEC.TAKEN_DIRECT_NEAR_CALL**
Taken speculative and retired direct near calls

**BR_INST_EXEC.TAKEN_INDIRECT_JUMP_NON_CALL_RET**
Taken speculative and retired indirect branches excluding calls and returns

**BR_INST_EXEC.TAKEN_INDIRECT_NEAR_CALL**
Taken speculative and retired indirect calls

**BR_INST_EXEC.TAKEN_INDIRECT_NEAR_RETURN**
Taken speculative and retired indirect branches with return mnemonic

**BR_INST_RETIRED.ALL_BRANCHES**
All (macro) branch instructions retired.

**BR_INST_RETIRED.ALL_BRANCHES_PS**
All (macro) branch instructions retired.

**BR_INST_RETIRED.CONDITIONAL**
Conditional branch instructions retired.
Continuous system-wide profiling of
High Performance Computing applications

BR_INST_RETIRED.CONDITIONAL_PS
Conditional branch instructions retired.

BR_INST_RETIRED.FAR_BRANCH
Far branch instructions retired.

BR_INST_RETIRED.NEAR_CALL
Direct and indirect near call instructions retired.

BR_INST_RETIRED.NEAR_CALL_PS
Direct and indirect near call instructions retired.

BR_INST_RETIRED.NEAR_CALL_R3
Direct and indirect macro near call instructions retired (captured in ring 3).

BR_INST_RETIRED.NEAR_CALL_R3_PS
Direct and indirect macro near call instructions retired (captured in ring 3).

BR_INST_RETIRED.NEAR_RETURN
Return instructions retired.

BR_INST_RETIRED.NEAR_RETURN_PS
Return instructions retired.

BR_INST_RETIRED.NEAR_TAKEN
Taken branch instructions retired.

BR_INST_RETIRED.NEAR_TAKEN_PS
Taken branch instructions retired.

BR_INST_RETIRED.NOT_TAKEN
Not taken branch instructions retired.

BR_MISP_EXEC.ALL_BRANCHES
Speculative and retired mispredicted macro conditional branches

BR_MISP_EXEC.ALL_CONDITIONAL
Speculative and retired mispredicted macro conditional branches

BR_MISP_EXEC.ALL INDIRECT_JUMP_NON_CALL_RET
Mispredicted indirect branches excluding calls and returns

BR_MISP_EXEC.NONTAKEN_CONDITIONAL
Not taken speculative and retired mispredicted
Continuous system-wide profiling of
High Performance Computing applications

macro conditional branches

BR_MISP_EXEC.TAKEN_CONDITIONAL
Taken speculative and retired mispredicted macro conditional branches

BR_MISP_EXEC.TAKEN_INDIRECT_JUMP_NON_CALL_RET
Taken speculative and retired mispredicted indirect branches excluding calls and returns

BR_MISP_EXEC.TAKEN_INDIRECT_NEAR_CALL
Taken speculative and retired mispredicted indirect calls

BR_MISP_EXEC.TAKEN_RETURN_NEAR
Taken speculative and retired mispredicted indirect branches with return mnemonic

BR_MISP_RETIRED.ALL_BRANCHES
All mispredicted macro branch instructions retired.

BR_MISP_RETIRED.ALL_BRANCHES_PS
Mispredicted macro branch instructions retired.

BR_MISP_RETIRED.CONDITIONAL
Mispredicted conditional branch instructions retired.

BR_MISP_RETIRED.CONDITIONAL_PS
Mispredicted conditional branch instructions retired.

BR_MISP_RETIRED.NEAR_TAKEN
number of near branch instructions retired that were mispredicted and taken.

BR_MISP_RETIRED.NEAR_TAKEN_PS
number of near branch instructions retired that were mispredicted and taken.

MEM_LOAD_UOPS_LLHCI_HIT RETIRED.XSNP_HIT
Retired load uops which data sources were LLC and cross-core snoop hits in on-pkg core cache.

MEM_LOAD_UOPS_LLHCI_HIT RETIRED.XSNP_HITM
Retired load uops which data sources were HitM responses from shared LLC.

MEM_LOAD_UOPS_LLHCI_HIT RETIRED.XSNP_HITM_PS
Retired load uops which data sources were HitM responses from shared LLC.

MEM_LOAD_UOPS_LLHCI_HIT RETIRED.XSNP_HIT_PS
Continuous system-wide profiling of High Performance Computing applications

Retired load uops which data sources were LLC and cross-core snoop hits in on-pkg core cache.

MEM_LOAD_UOPS LLC_HIT RETIRED.XSNP_MISS
Retired load uops which data sources were LLC hit and cross-core snoop missed in on-pkg core cache.

MEM_LOAD_UOPS LLC_HIT RETIRED.XSNP_MISS_PS
Retired load uops which data sources were LLC hit and cross-core snoop missed in on-pkg core cache.

MEM_LOAD_UOPS LLC_HIT RETIRED.XSNP_NONE
Retired load uops which data sources were hits in LLC without snoops required.

MEM_LOAD_UOPS LLC_HIT RETIRED.XSNP_NONE_PS
Retired load uops which data sources were hits in LLC without snoops required.

MEM_LOAD_UOPS LLC_MISS RETIRED.LOCAL_DRAM
Retired load uops which data sources missed LLC but serviced from local dram.

MEM_LOAD_UOPS RETIRED.HIT_LFB
Retired load uops which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.

MEM_LOAD_UOPS RETIRED.HIT_LFB_PS
Retired load uops which data sources were load uops missed L1 but hit FB due to preceding miss to the same cache line with data not ready.

MEM_LOAD_UOPS RETIRED.L1_HIT
Retired load uops with L1 cache hits as data sources.

MEM_LOAD_UOPS RETIRED.L1_HIT_PS
Retired load uops with L1 cache hits as data sources.

MEM_LOAD_UOPS RETIRED.L1_MISS
Retired load uops which data sources following L1 data-cache miss

MEM_LOAD_UOPS RETIRED.L1_MISS_PS
Retired load uops which data sources following L1 data-cache miss.

MEM_LOAD_UOPS RETIRED.L2_HIT
Retired load uops with L2 cache hits as data sources.
Continuous system-wide profiling of High Performance Computing applications

MEM_LOAD_UOPS_RETIRED.L2_HIT_PS
Retired load uops with L2 cache hits as data sources.

MEM_LOAD_UOPS_RETIRED.L2_MISS
Miss in mid-level (L2) cache. Excludes Unknown data-source.

MEM_LOAD_UOPS_RETIRED.L2_MISS_PS
Retired load uops with L2 cache misses as data sources.

MEM_LOAD_UOPS_RETIRED.LLC_HIT
Retired load uops which data sources were data hits in LLC without snoops required.

MEM_LOAD_UOPS_RETIRED.LLC_HIT_PS
Retired load uops which data sources were data hits in LLC without snoops required.

MEM_LOAD_UOPS_RETIRED.LLC_MISS
Miss in last-level (L3) cache. Excludes Unknown data-source.

MEM_LOAD_UOPS_RETIRED.LLC_MISS_PS
Miss in last-level (L3) cache. Excludes Unknown data-source.

CYCLE_ACTIVITY.STALLS_L1D_PENDING
Execution stalls due to L1 data cache misses

CYCLE_ACTIVITY.STALLS_L2_PENDING
Execution stalls due to L2 cache misses.

CYCLE_ACTIVITY.STALLS_LDM_PENDING
Execution stalls due to memory subsystem.

CYCLE_ACTIVITY.CYCLES_NO_EXECUTE
Total execution stalls