2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)

Analyzing the Impact of Operating System Activity of different Linux Distributions in a Distributed Environment

Roberto Giorgi[1], Marco Procaccini[1] and Farnam Khalili[1],[2]

[1]Department of Information Engineering and Mathematics - University of Siena
[2]Department of Information Engineering - University of Florence
{giorgi,procaccini,khalili}@diism.unisi.it

***Abstract*—A rise in the number of threads in large-scale applications running on multi-node architectures makes operating system activity increasingly more relevant. Therefore, evaluation methodologies need to account for these activities. We decided to build our evaluation environment through the COTSon simulator. Moreover, our environment permits flexible Design Space Exploration (DSE) by making easy the management of many experiments and the characterizations of Operating System (OS) activity.**

In this paper, we show the result analysis tool flow and the OS impact of different Linux distributions running on a distributed environment consisting of several nodes with a full OS. In order to quantify our results, we use matrix multiplication benchmark executed through a DataFlow model, named DataFlow-Threads(DF-Threads). We analyze key metrics like L2 cache miss rate, execution cycles, data access latency, and kernel cycles showing up to 60% performance variations among the different OS distributions.

***Index Terms*—Performance evaluation, Computer architecture, Computer simulation, Matrix Multiply, Distributed computing, High performance computing, Operating systems**

I. Introduction

Nowadays the size of high-performance computers (HPCs) is growing to thousands of nodes and almost all top supercomputers deploy Linux on each node. In order to achieve concurrency with asynchronous parallelization of jobs, an operating system (OS) may be responsible to distribute the application across the available hardware resources. In that sense, the mechanisms like inter-process communication, data synchronization, and thread/process management are all parts of OS responsibility [1]. Hence, part of system performance is consumed by OS to schedule and manage the hardware resources. Meanwhile, system performance began to be degraded in large-scale applications using hundreds to thousands of nodes due to the time gaps between speedy processes and lagging ones [2]–[4]. This degradation arises from the variability due to OS impact for each node system activity, which is predominantly irrelevant from the parallel application.

In Design Space Exploration (DSE), understanding the OS impact on system performance while running a certain application is crucial. Constantly growing complexity of multi-core multi-node architectures [5], and ever-increasing powerful applications, make full system evaluation with hardware prototyping or even at FPGA-based setups quite challenging and time-consuming. As such, in order to evaluate, explore, and improve time-to-market, simulators are significant portions of modern performance evaluation methodologies. In [6], [7] COTSon [8], a full system simulator, has been leveraged in order to provide Distributed Scheduler to support many-node architectures, and significantly improve scalability and power estimation. However, in these works the OS impact on performance still remained uncertain.

In this paper, we present a set of experiments that we performed by the COTSon simulator [8] to analyze the OS activity. In order to automatically perform the evaluation through the simulator, we designed our DSE toolset, which allows us to model those aspects of design that are not yet available on the market and simulate multi-node architectures as well. Thanks to our DSE toolset [9], we were able to easily evaluate the OS impact whereas we launched a novel execution model based on DataFlow paradigm called DataFlow-Threads (DF-Threads) [10]–[12].

We leveraged our tools in the context of AXIOM project in order to explore hot case studies like smart home and video surveillance [13], [14]. While our previous research revolved around the execution model [15], in this work we focus on the operating system to understand how to extract the OS impact on performance, and how much is the contribution of kernel space and user space. To quantify the result, we run a matrix multiplication benchmark on top of the DF-Threads execution model. Then, we extract important standard metrics like L2 cache miss Rate, memory activity, kernel activity and execution time for several Linux distribution while the number of nodes increases.

The remainder of this paper is structured as follows: in Section II, we discuss and compare the important features of different simulators, and why we chose COTSon simulator; in Section III, we show our DSE toolset and methodology and how to analyze the OS activity; in Section IV, we evaluate the

2377-5750/19/$31.00 ©2019 IEEE
DOI 10.1109/PDP.2019.00068

impact of several Linux distribution in a multi-node distributed system based; in Section V, we highlight some related work studying the OS impact on performance, and finally, we conclude the paper in Section VI.

II. Toolset Feature Comparison

Analyzing the OS activity of a multi-node architecture, even at the prototype stage, might not be straightforward. Even preparing a complete setup and hardware prototyping might impose several limitations. In this section, we briefly highlight the reasons why we chose a simulator like COTSon [8], how our DSE toolset facilitates extracting the intervention of OS on performance.

Prototyping the architecture with physical hardware is quite costly and inflexible towards any updates of Instruction Set Architecture (ISA). In case of FPGA-based setups, the hardware and software must be configured and well set up, which demands remarkable effort and time. In case of simulators, they might not show satisfying credibility, but since they evolve, they also improve in terms of many features like reliability. Simulators are useful to achieve a certain level of accuracy, speed and scalability without any considerable cost. There are other parameters such as easy observability, and reproducibility which make simulators preferable in comparison with a physical implementation.

There are different features to classify and characterize a simulator (some with overlapping). We report some of those features in Table I, to compare several simulators. According to whether the OS action is included or not, there are full-system simulators and user-space simulators. All listed simulator here are examples of full-system simulators. Considering if they can execute in parallel or not, we have sequential simulators and parallel ones. While the first ones are more accurate [25], the required time for simulation takes considerably if the complexity of the platform under simulation increases. On the other side, the parallel simulators, like COTSon [8], are able to take advantage of multiple processing cores of the host by launching virtual machines, where each of them represents a node in a distributed architecture.

SimFlex and Gems both rely on Intel's Simics which is an off-the-shelf sequential emulator to model functional as well as their own models for memory hierarchy and interactions of cores. GEMS has much lower performance than COTSon due to the poor performance of their timing model part, which relies on timing-first simulation approach. SimFlex relies on statistical sampling approach to increase the simulation rate, but it is not able to observe its entire characteristics.

Graphite has been developed at MIT and is based on the PIN binary instrumentation package (i.e., a functional trace-driven simulator). Among all the multi-core simulators listed in table I, only for COTSon and Graphite there are results regarding simulation of 1000 cores [24], [26]–[28].

Among listed simulators in Table I, only COTSon and SimFlex support fast-forwarding technique, which is a feature of functional mode in order to accelerate simulation. One of the common technique to accelerate the simulation is sampling, which selects some instructions simulating in cycle-accurate mode (timing mode). Cycle-accurate simulations last a long time, there are therefore lots of researches on how to enhance the cycle-accurate simulation rate.

Moreover, through simulators like COTSon, we can easily extend the ISA [29], [30], and implement any types of complicated architectures suitable for any execution models like DF-Threads [10]–[12] launched on a multi-node architecture. For example, in order to measure the performance of hundreds of multi-core, multiprocessor nodes in an affordable amount of time, we should be able to have a full-system simulator supporting full software stack, memory hierarchy, application benchmarks and all devices.

Among well-known simulators, those which can support a full system simulation (i.e., including OS) are beneficial since they provide us to analyze and observe the OS activity and contribution on performance. As [2], [3] discovered, the performance of a large-scale application running OS, began to drop down as number of nodes increases. However, in order to check the OS intervention, specifically, in scaling up the cluster, a complete simulation ranging from multi-core nodes up to full clusters with complete network simulation is unconditionally needed. COTSon is relatively mature, and can support both scalability and running OS, permits us to extract kernel activities while running a multi-node application. Moreover, it is composed of a set of pluggable component, in which most extensions can be featured to allow us to benchmark execution models like DF-Threads, meantime, stream parameters like kernel cycle.

III. Methodology

In this section, we describe the methodology and the software toolset adopted to perform the analysis of the activity of different Linux distributions in a multi-core/multi-node environment.

We rely on a full system simulation framework based on a customized version of the COTSon simulator and a set of Design Space and Exploration (DSE) tool, which help us collect and analyze the results.

A. COTSon Simulator

COTSon [8] is a simulator, which performs a full-system simulation decoupling the functional execution from the timing behavior ("functional approach"). The functional model is executed inside the SimNow virtualizer, a tool used by AMD to test their CPU and platform. Therefore, we are able to run off-the-shelf Linux distribution, modeling a realistic situation like interrupts, exceptions, virtual memory management and etc. Moreover, new functionalities can be modeled and designed defining timing models for

TABLE I: Comparison different presented features in simulators supporting OS and multi-core simulation of large-scale applications

Simulator Name	Parallel/Sequential simulation	Simulator Platform	Fast-forwarding	Sampling support
COTSon [8]	Parallel	Software-based, Virtual Machine	x	x
GEMS [16]	Sequential	Virtual Machine		
ProtoFlex [17]	Parallel	FPGA-based		
RAMP [18]	Parallel	FPGA-based		
SimFlex [19]	Sequential	Software-based, Virtual Machine	x	x
SimNow [20]	Sequential	Software-based		
Simics [21]	Sequential	Virtual Machine		
SimOS [22]	Sequential	Software-based		
Trace Factory [23]	Sequential	Software-based		
Graphite [24]	Sequential	Software-based		

each possible component of the architecture (i.e., CPU, caches, network switch). As depicted in Table II, we modeled the key parameter of the cores of the SimNow virtual machine.

The simulator gives us the possibility to model a complete network infrastructure with several topologies (i.e., star, mesh, torus), allowing us to build a distributed system with many-core/many-nodes (see Figure 1).

Furthermore, we have the possibility to execute different well-known programming models (i.e., Cilk++, OpenMPI, DSM) and extend COTSon to support the DF-Threads execution model [10] and the Distributed Thread Scheduler [11], on which DF-Threads relies.

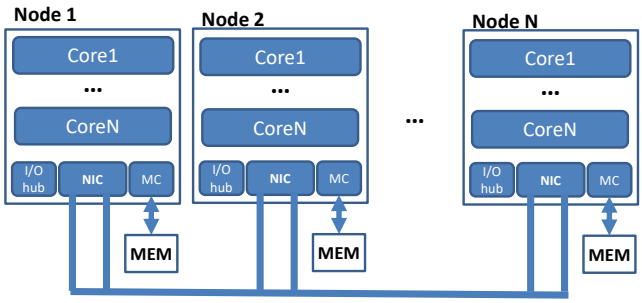

Fig. 1: Distributed System architecture with many-cores/many-node made into the COTSon simulator. Each one is modeled as System-on-Chip (SoC). MC=Memory Controller. Network interface controller (NIC) models the network behavior.

B. *DSE Tools*

To easily manage the simulation framework, run several experiments, analyze the result and guarantee a proper scientific methodology, we developed a set of tool for the Design Space and Exploration (DSE). As described in Figure 2, we can prepare the simulation environment through the "MYINSTALL" program, taking care of packages dependencies, Linux distributions and which type of simulator we need (i.e, COTSon). The MYINSTALL allows us to enable multiple user on the host machine to run configuration of their own simulation setup without the need of system administrator intervention. Moreover, the MYINSTALL tool automatically performs several regression tests at the end of the installation phase, in order to verify the correctness of the installed software. Thanks to this tool, we are able to install the complete environment in less than 10 minutes, saving hours of work.

TABLE II: Multi-core architectural parameters.

Parameter	Description
SoC	1-core connected by a shared-bus, IO-hub, MC, high-speed transceivers
Core	3GHz, in-order super-scalar
Branch Predictor	two-level (history length=14bits, pattern-history table=16kB, 8-cycle miss-prediction penalty)
L1 Cache	Private I-cache 32 KB, private D-cache 32 KB, 2 ways, 3-cycle latency
L2 Cache	Private 2,8,32,256KB, 4 ways, 5-cycle latency
L3 Cache	Shared 4MB, 4 ways, 20-cycle latency
Coherence protocol	MOESI
Main Memory	1 GB, 100 cycles latency
I-L1-TLB, D-L1-TLB	64 entries, full-associative, 1-cycle latency
L2-TLB	512 entries, direct access, 1-cycle latency
Write/Read queues	200 Bytes each, 1-cycle latency

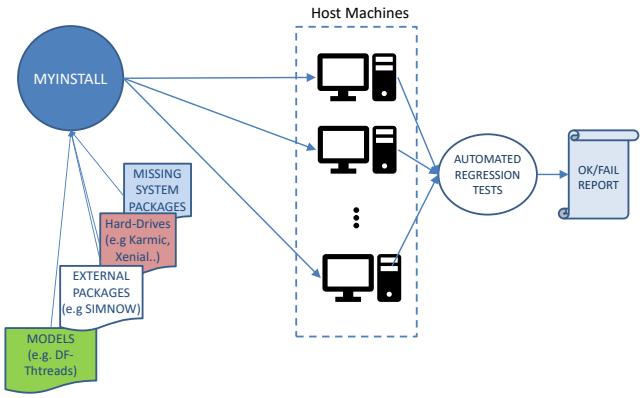

Fig. 2: TOOLFLOW for the MYINSTALL tool. MYINSTALL prepares the whole environment for simulation-based Design Space Exploration with a single command among several host machines. At the end of each installation, an automated regression test is performed.

To provide an operating system (OS) for the SimNow virtual machine, we provide the GEN-IMAGE tool, through

which is possible the easily generation of several types of customized OS versions. As we can see in Figure 3, the creation of a new hard-drive image (system image) based on a Linux Official public distribution relies on the Debootstrap tool. Furthermore, the GENIMAGE tool generates the hardware description file (BSD file) and the necessary BIOS/ROM for the virtual machine (i.e., AMD BIOS/ROM for SimNow). At the end of the image generation process, the tool performs an automated validation test, using the Tesseract image recognition package, to check the screen output .

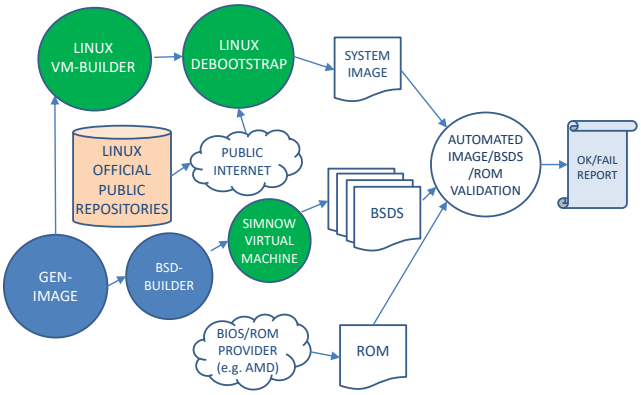

Fig. 3: TOOLFLOW for the GENIMAGE tool. The tool generates the hard drive image (system image) and the BSD (hardware description) based on the user requirements. A validation test is made at the end of the workflow, through the Tesserac image recognition package of the screen.

The ”MYDSE” tool is responsible to manage the experiment loop, configure the simulator with the specified simulation point and collect several generated files of the simulator in an ordered way (see Figure 4). The usefulness of this tool relies on the usage of a simple text file with a ”$\langle key \rangle$=$\langle list\_values \rangle$” syntax, named INFOFILE, through which is possible to easily define several simulation configurations.

For example, we are able to specify different GLIBC library versions into the INFOFILE, having the possibility to test multiple operating systems, with a different version of GLIBC library, in a complete automatic way.

During each simulation loop, the tool handles possible failures or errors, providing a timeout mechanism in order to retry killed simulations. The timeout is adaptive and depends on a simulation estimation model (i.e, proportional to the number of cores, number of nodes).

To be able to analyze the huge amount of data produced by the experiments without losing the accuracy, we designed the GTCOLLECT and the GTRAPH tools. The GTCOLLECT tool is responsible to gather all the information regarding a specific experiment and organizes the results in a tabular layout. Furthermore, the tool permits the analysis of multiple experiment repetitions (at least 5 times to have the classical ”statistically stable” number of measurements), computes the average and other statistics, like Standard Deviation, on the dataset. Finally, the GTGRAPH tool is able to derive a graphical view of the results to easily analyze them.

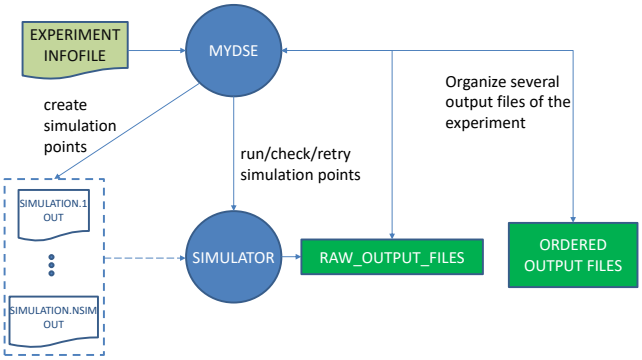

Fig. 4: TOOLFLOW of the MYDSE tool. The tool creates/runs/checks and retries several simulation points defined into the experiment INFOFILE. At the end of the experiment loop, MYDSE tool organizes the raw output file produced by simulator in a better way, in order to be easily analyzed by other tools.

C. Kernel cycles extraction

As we described in section III-A, COTSon uses a functional approach for the simulation, giving us the possibility to model the system features through the specification of the timing models. In order to extract the kernel activity, we defined a timing model for a CPU, x86_64 instruction set based, where we identify the kernel instructions analyzing the memory address accessed by each instruction. As depicted in Figure 5, we analyze the most significant bit of memory address accessed by the current instruction and if the bit is equal to 1, it means that the instruction is trying to read or write in the Kernel space memory section. Consequently, we mark the instruction as a kernel type. Finally, we integrated this proceeding in the CPU timing model of COTSon.

IV. Experiments

In this section, we present the tests performed into a distributed simulation environment in order to study the influence of several Linux distribution on the performance. For the comparison, we selected the DF-Threads execution model and the Matrix Multiplication Benchmark. We analyzed how the execution time differs varying the Operating System. Moreover, we studied key aspects (i.e., L2 Miss Rate, Kernel Cycles) to evaluate the impact of each distribution on the execution time.

We configured the distributed simulation environment with a node range from 1 to 4 and a core range from 1 to 32. The input sizes used for the Matrix Multiplication benchmark vary from 64 to 1024.

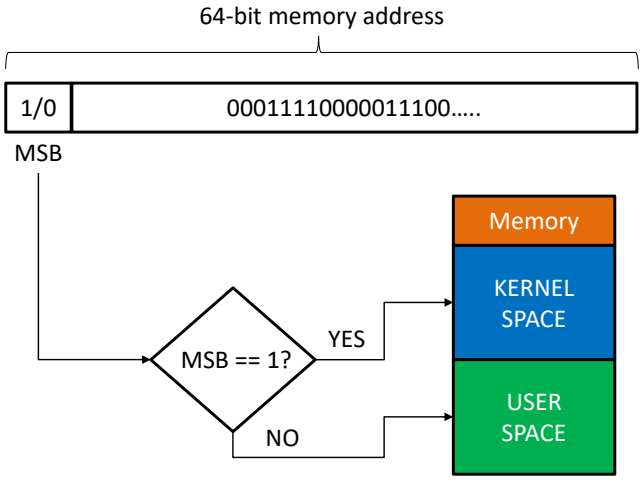

Fig. 5: Kernel instruction extracting technique in COTSon simulator. The Most Significant Bit (MSB) of the memory address accessed by the current instruction is analyzed and if the MSB is equal to 1, it means that the instruction is trying to access the kernel space and it is a Kernel Instruction.

A. Operating system distributions

The operating system are based on the Linux distribution Ubuntu like and we chose four different kernel versions to perform our tests:

- Karmic: it is the Ubuntu 9.10 LTS version. The distribution focuses on improvements in cloud computing as well as further improvements in boot speed.
- Tfx: it represents the Maverick version of the Ubuntu kernel (version 10.10)
- Trusty (Ubuntu 14.04 LTS): the main improvements were based on increasing the performance and the maintainability.
- Xenial: It is the Ubuntu 16.04 LTS version

B. Cache sensitivity

The first comparison between the Linux distribution regards the execution time and the scalability. We vary the L2 cache size with small sizes (from 2 to 32 KiB), keeping fixed the input size (matrix size=512) and the cores number. As we can see in Figure 6, the Trusty distribution obtains better results, both in execution time and scalability, than the other releases, outperforming the Xenial version by a factor of 60% in the one node execution. This variation rises from the different configurations and daemons included in the distro. Moreover, the Karmic distribution showed a quite similar result as the Trusty when we increase the number of nodes, confirming the improvements in cloud computing and performances made by the Linux development communities. In all cases, a larger cache size improves the performance as expected. Once we increase the number of nodes, one interesting effects is also that the total capacity of caches will increase. Moreover, we observe a reasonable scaling of performance with the number of nodes.

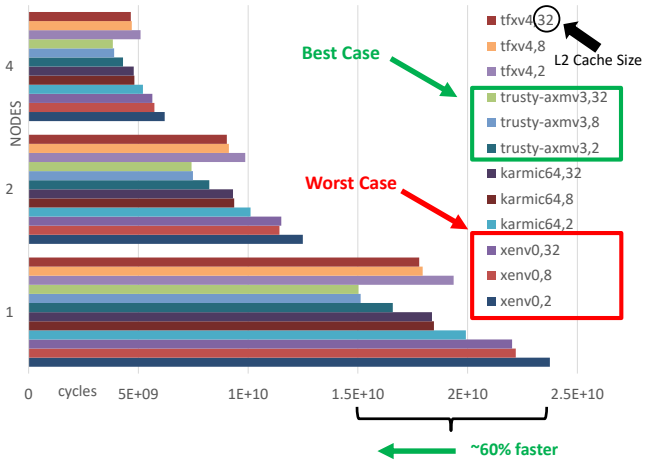

Fig. 6: Study of the performance variation (cycles) as we vary the L2 cache size (from 2 to 32KiB) and the operating system distribution. In the case of one node, this variation is up to 60% due to the different types of daemons and configurations included in distros.

C. Matrix size sensitivity

In the following experiment, we keep fixed the L2 cache size (256 KiB) and we vary the number of nodes and the matrix size from 64 to 1024. As depicted in Figure 7 and Figure 8, there is a strong correlation between the data access latency seen by the processor (Figure 7). Moreover, the miss rate of the L2 cache (Figure 8), shows that the L2 cache size has a strong impact on the performance and therefore it may plays a crucial role in any operating system distribution observed.

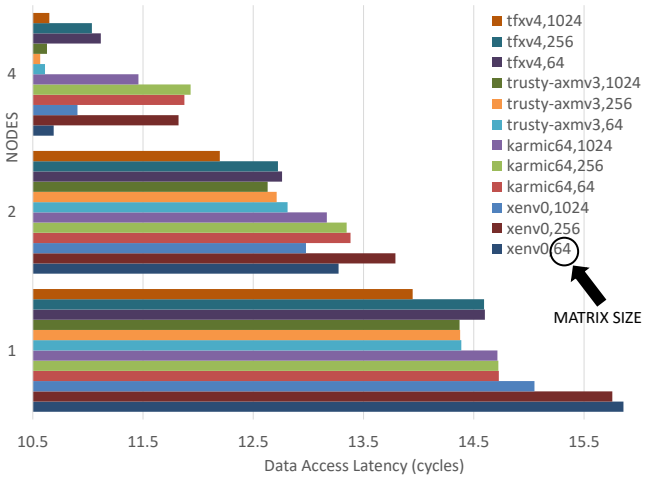

Fig. 7: Data access latency seen by the processor varying the input size (from 64 to 1024), the operating system distribution and the number of nodes.

Again, we keep fixed the L2 cache (256 KiB), analyzing the effect of different matrix size (from 64 to 1024) and therefore a different number of generated threads. As can be seen from Figure 9, we observe a certain variation in the

performance with the OS distribution and the Trusty release seems the best among the tested ones in most cases. Also in this case, we obtain a quite good scalability factor increasing the input size and the number of nodes.

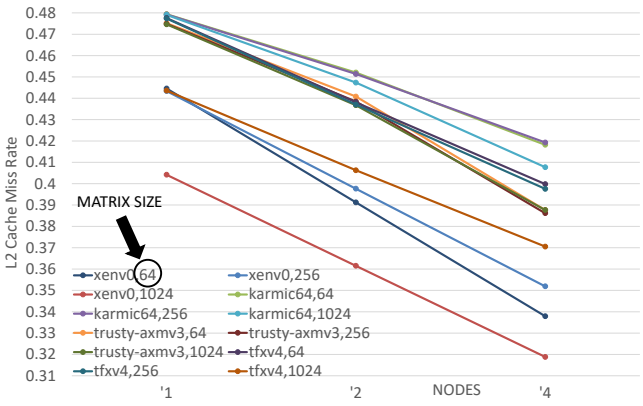

Fig. 8: Analysis of the L2 Cache Miss Rate as varying the number of nodes, the operating system distribution and the input size (from 64 to 1024).

D. *Kernel activity*

Regards to study the influence of the kernel execution time on the total cycles of the application, we configured the experiment keeping fixed the L2 cache (256 KiB), varying the matrix input size (from 64 to 1024), the number of nodes (from 1 to 4) and the OS distributions. As we can observe in Figure 10, the Karmic distribution shows better results, keeping the percentage of the kernel cycles lower than the others releases in any configuration. Generally, the kernel impact on the total cycles is proportional to the number of nodes, implying that the influence of the kernel is more important for multi-node configurations.

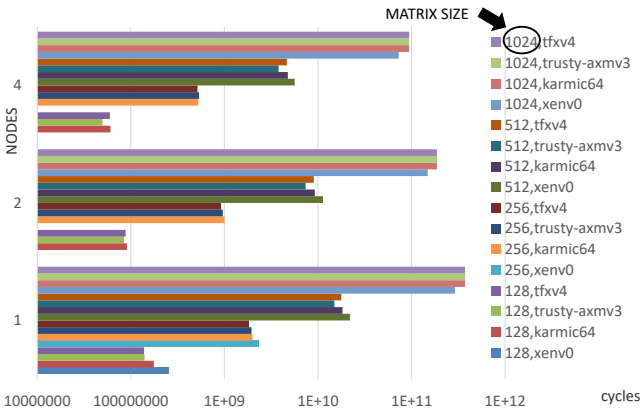

Fig. 9: Variation in number of execution cycles in different operating system distribution as we vary the input size (matrix size = 128, 256, 512, and 1024). The trusty-axmv3 seems to have the best performance compared to the other distribution.

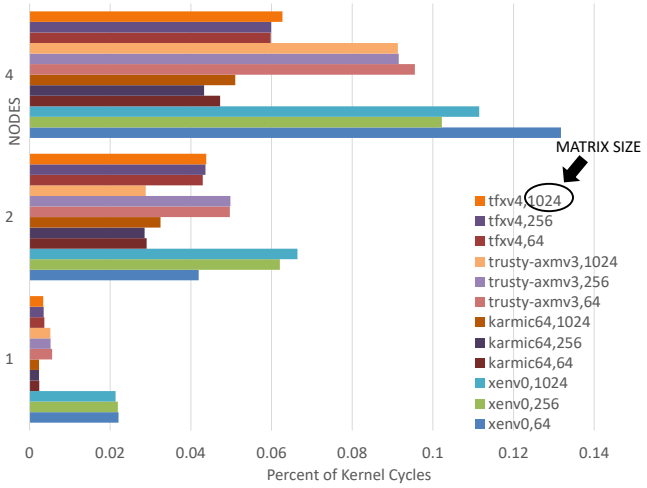

Fig. 10: The percentage of kernel cycles on the input size from 64 to 1024. The kernel time may occupy from few percent (1 Node execution) to more than 13% of the total time (4 Nodes execution).

V. Related Work

Investigating on the kernel activity and system impact in parallel computing has a long story. Our contribution includes for the first time the exploring and quantifying the influence of OS kernel in a novel execution model (DF-Threads [10]) on a large-scale multi-node system.

Many researchers have explored several impact on performance variations in large-scale parallel computers. For instance, Petrini et al. [2] propose a performance-analysis methodology including those artifacts that degrade application performance, and determines which sources of noise have the most impact on performance, and therefore those to eliminate. In [31], Betti et al. propose an extension to the Linux Kernel in a large-scale supercomputer to detect most time-consuming system activities, through which the OS noise can be reduced on a single node leading better scalability. Moreover, the design co-ordinates cluster level operations such as data checkpoint or debugging.

In [32], Ferreira et al. demonstrate the importance of how noise is generated, regarding frequency and duration, and how this impact changes with application scale using a kernel-based noise injection in a large-scale application. They also show how aspects of application, like computation/communication ratios, collective communication sizes, amplify or absorb OS noise. On the contrary, Wallas [33] shows the bigger and less frequent efficient task for small-benchmarks by simulating OS noise for a 3500+ node Cray XT3 machine. For more information regarding micro-benchmarks, Sottile et al. in [34] identify features within the noise due to frequent activities that are not part of user code and discuss different types of micro-benchmarks like Fixed Work Quantum (FWQ) and Fixed Time Quantum (FTQ). Cui et al. in [35], propose a benchmark to address the parallel scalability of OS on large-scale multi-core platforms. They evaluate that all micro-benchmarks in their design scale bad, except dupbench. Their analysis of kernel

activity indicates that kernel synchronization primitives are the main reason of the poor scalability.

Tsafrir et al. [36] show the complexity of implementation of global tick synchronization in a large-scale application and how this can degrade the performance. Moreover, they present how the OS impact is linearly proportional to the size of the cluster under certain condition and considering probabilistic arguments. They identify one of the major impact which indirectly imposes an extra overhead to OS Clock Interrupts, so-called ticks. To cope with this problem, they propose a smart–tick to merge events to reduce the number of the timer interrupt.

In [37], Mraz addresses issues regarding message passing variance when executing real-time parallel applications. The author proposes a method to reduce performance variation through kernel extensions, controlling interrupt level and process priorities as well as synchronization at runtime.

Similarly to our work, Wentzlaff discusses some of the problems regarding scalability in monolithic operating systems [38]. They propose a factored operating system targeting 1000+ core with space-sharing to improve scalability. In [39], Song performs experiments with four applications on multi-core architectures on which a proposed framework runs multiple commodity, providing the illusion of running on a single OS. The results show that their design has better scalability in comparison with Linux.

In [40], authors propose a full stack simulation system targeting heterogeneous kilo-core architectures, which is able to extend and add new instruction sets to support DataFlow-Threads (DF-Threads) execution model [10]. In the context of the AXIOM project [41]–[49], recent papers discuss a full system framework for a many-board embedded computing. We evaluated the AXIOM embedded system through real life applications such as Smart Video Surveillance(SVS) and Smart Home Living (SHL) [13], [14]. The work presented in this paper help us in the deeply analysis on how to improve the performance of such case-studies. In the context of the TERAFLUX project [6], [50], [51], we started to develop DSE tools in order to evaluate a complex architecture with e.g., 1000 general purpose cores and running a full OS like Linux. However, in recent papers, we did not analyze the impact of different OS distributions on performance. In this paper, we explore the important metrics while running several distributions of Linux and evaluate their impact on performance using our DSE toolset. We have shown in experiments, that we can vary architecture parameters like cache size and observe OS impact which is not possible in a fixed architecture.

VI. Conclusions

As related research has discovered, operating system (OS) increasingly affects the performance when the number of nodes increases. In order to study this impact, a simulator which can support both OS and multi-node simulation is valuable. We found that the OS impact could vary among different types of Linux distributions. In this paper, we compare several Linux Ubuntu distributions to understand the performance sensitivity when accounting for all the software components like run-time and OS.

Then, we explained our motivation for selecting COTSon simulator as one of our tool. We present our Design Space Exploration (DSE) toolset through which we easily manage and establish a distributed system environment of many-nodes. Our DSE toolset facilitates steps to setup simulated architecture on severals machines, spending less than ten minutes for each of them and with reduced human interaction. Thanks to our DSE toolset, we were able to extract a huge amount of data results, and their graphical representation, evaluate several Linux distributions on the simulated platform in a fast and easy way, saving hours of work.

As one benchmark of high interest nowadays is matrix multiplication (a fundamental operation, e.g., the Deep Neural Networks) and we chose it as a simple driver for our experiments in order to study the OS impact on performance, while varying key metrics of the architecture (e.g, L2 Cache size) and increasing the number of nodes of the distributed environment. We extract and analyze the key metrics like L2 Cache Miss Rate, execution cycles, data access latency, and kernel cycles for different Linux distribution. We showed the behavior of OS varying the L2 Cache size, reaching up to 60% in performance variations due to the different types of configurations and daemons installed. Moreover, with the same Linux distributions, the OS is responsible for a portion of the execution time which may exceed 13% in case of four nodes.

VII. Acknowledgment

The authors would like to thank the AXIOM H2020 project (id. 645496), the TERAFLUX (id. 249013), and the HiPEAC (id. 779656), and finally, the anonymous reviewers for their helpful comments.

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system concepts essentials*. John Wiley & Sons, Inc., 2014.

[2] F. Petrini, D. J. Kerbyson, and S. Pakin, "The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q," in *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, (New York, NY, USA), pp. 55–, ACM, 2003.

[3] P. Terry, A. Shan, and P. Huttunen, "Improving application performance on hpc systems with process synchronization," *Linux J.*, vol. 2004, pp. 3–, Nov. 2004.

[4] T. Jones, S. Dawson, R. Neely, W. Tuel, L. Brenner, J. Fier, R. Blackmore, P. Caffrey, B. Maskell, P. Tomlinson, and M. Roberts, "Improving the scalability of parallel jobs by adding parallel awareness to the operating system," in *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, (New York, NY, USA), pp. 10–, ACM, 2003.

[5] S. Borkar and A. A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, 2011.

[6] R. Giorgi, R. Badia, *et al.*, "TERAFLUX: Harnessing dataflow in next generation teradevices," *Microprocessors and Microsystems*, vol. 38, no. 8, Part B, pp. 976–990, 2014.

[7] A. Portero, Z. Yu, and R. Giorgi, "Teraflux: Exploiting tera-device computing challenges," *ELSEVIER Procedia Computer Science*, vol. 7, pp. 146–147, 2011. Proc. 2nd European Future Technologies Conf. and Exhibition 2011 (FET 11).

[8] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTSon: infrastructure for full system simulation," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 52–61, 2009.

[9] R. Giorgi, M. Procaccini, and F. Khalili, "A design space exploration tool set for future tera-scale high-performance computers," in *11th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO)*, January 2019.

[10] R. Giorgi and P. Faraboschi, "An introduction to DF-Threads and their execution model," in *IEEE MPP*, (Paris, France), pp. 60–65, Oct. 2014.

[11] R. Giorgi and A. Scionti, "A scalable thread scheduling co-processor based on data-flow principles," *Future Generation Computer Systems*, vol. 53, pp. 100–108, Dec. 2015.

[12] L. Verdoscia and R. Giorgi, "A data-flow soft-core processor for accelerating scientific calculation on FPGAs," *Mathematical Problems in Engineering*, vol. 2016, pp. 1–21, Apr. 2016. article ID 3190234.

[13] R. Giorgi, N. Bettin, P. Gai, X. Martorell, and A. Rizzo, *AXIOM: A Flexible Platform for the Smart Home*, ch. 3, pp. 57–74. Cham: Springer Int.l Pub., 2016.

[14] C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, *et al.*, "The AXIOM software layers," in *IEEE Proc. 18th EUROMICRO-DSD*, pp. 117–124, Aug. 2015.

[15] R. Giorgi, "Scalable embedded computing through reconfigurable hardware: comparing df-threads, cilk, OpenMPI and jump," *ELSEVIER Microprocessors and Microsystems*, vol. 63, pp. 66–74, Aug. 2018.

[16] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[17] E. S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, K. Mai, and B. Falsafi, "Protoflex: Towards scalable, full-system multiprocessor simulations using fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, pp. 15:1–15:32, June 2009.

[18] G. Gibeling, A. Schultz, and K. Asanovic, "The ramp architecture & description language," in *2nd Workshop on Architecture Research using FPGA Platforms*, 2006.

[19] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "Simflex: statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, 2006.

[20] O. A. Place, "Amd simnow™ simulator," 2004.

[21] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[22] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "Complete computer system simulation: The simos approach," *IEEE Parallel & Distributed Technology: Systems & Applications*, vol. 3, no. 4, pp. 34–43, 1995.

[23] R. Giorgi, C. Prete, G. Prina, and L. Ricciardi, "A workload generation environment for trace-driven simulation of shared-bus multiprocessor," in *IEEE Proc. 30th Hawaii Int.l Conf. on System Sciences (HICSS-30)*, vol. 1, (Maui, Hawaii), pp. 266–275, Jan. 1997.

[24] J. E. Miller, H. Kasture, *et al.*, "Graphite: A distributed parallel simulator for multicores," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pp. 1–12, IEEE, 2010.

[25] N. L. Binkert *et al.*, "The m5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, 2006.

[26] D. Vantrease *et al.*, "Corona: System implications of emerging nanophotonic technology," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, (Washington, DC, USA), pp. 153–164, IEEE Computer Society, 2008.

[27] M. Monchiero, J. H. Ahn, A. Falcón, D. Ortega, and P. Faraboschi, "How to simulate 1000 cores," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 10–19, 2009.

[28] R. Giorgi, "Exploring future many-core architectures: The TERAFLUX evaluation framework," in *Advances in Computers*, Advances in Computers, pp. 33–72, Elsevier, 2017.

[29] A. Portero, Z. Yu, and R. Giorgi, "T-star (t*): An x86-64 isa extension to support thread execution on many cores," in *HiPEAC ACACES-2011*, (Fiuggi, Italy), pp. 277–280, July 2011. poster.

[30] N. Ho, A. Portero, M. Solinas, A. Scionti, A. Mondelli, P. Faraboschi, and R. Giorgi, "Simulating a multi-core x86-64 architecture with hardware isa extension supporting a data-flow execution model," in *IEEE Proc. AIMS-2014*, (Madrid, Spain), pp. 264–269, Nov. 2014.

[31] E. Betti, M. Cesati, R. Gioiosa, and F. Piermaria, "A global operating system for hpc clusters," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pp. 1–10, IEEE, 2009.

[32] K. B. Ferreira, P. Bridges, and R. Brightwell, "Characterizing application sensitivity to os interference using kernel-level noise injection," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, p. 19, IEEE Press, 2008.

[33] D. Wallace, "Compute node linux: Overview, progress to date, and roadmap," in *Proceedings of the 2007 Cray User Group Annual Technical Conference*, 2007.

[34] M. Sottile and R. Minnich, "Analysis of microbenchmarks for performance tuning of clusters," in *Cluster Computing, 2004 IEEE International Conference on*, pp. 371–377, IEEE, 2004.

[35] Y. Cui, Y. Chen, and Y. Shi, "Osmark: A benchmark suite for understanding parallel scalability of operating systems on large scale multi-cores," in *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pp. 313–317, IEEE, 2009.

[36] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick, "System noise, os clock ticks, and fine-grained parallel applications," in *Proceedings of the 19th Annual International Conference on Supercomputing*, ICS '05, (New York, NY, USA), pp. 303–312, ACM, 2005.

[37] R. Mraz, "Reducing the variance of point to point transfers in the ibm 9076 parallel computer," in *Supercomputing'94., Proceedings*, pp. 620–629, IEEE, 1994.

[38] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): The case for a scalable operating system for multicores," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 76–85, Apr. 2009.

[39] X. Song, H. Chen, R. Chen, Y. Wang, and B. Zang, "A case for scaling applications to many-core with os clustering," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (New York, NY, USA), pp. 61–76, ACM, 2011.

[40] A. Portero, A. Scionti, Z. Yu, P. Faraboschi, C. Concatto, L. Carro, A. Garbade, S. Weis, T. Ungerer, and R. Giorgi, "Simulating the future kilo-x86-64 core processors and their infrastructure," in *45th Annual Simulation Symp. (ANSS12)*, (Orlando, FL), pp. 62–67, Mar 2012.

[41] R. Giorgi, M. Procaccini, and F. Khalili, "AXIOM: A scalable, efficient and reconfigurable embedded platform," in *Design, Automation and Test in Europe, the european event for electronic system design and test (DATE)*, March 2019.

[42] R. Giorgi, F. Khalili, and M. Procaccini, "Energy efficiency exploration on the zynq ultrascale+," in *The 30th International Conference on Microelectronics (ICM)*, December 2018.

[43] D. Theodoropoulos *et al.*, "The AXIOM project (agile, extensible, fast i/o module)," in *IEEE Proc. 15th Int.l Conf. on Embedded Computer Systems: Architecture, MOdeling and Simulation*, pp. 262–269, July 2015.

[44] R. Giorgi, S. Mazumdar, S. Viola, P. Gai, S. Garzarella, *et al.*, "Modeling multi-board communication in the axiom cyber-physical system," *Ada User Journal*, vol. 37, pp. 228–235, December 2016.

[45] P. Burgio, C. Alvarez, E. Ayguade, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, N. Navarro, and R. Giorgi, "Simulating next-generation cyber-physical computing platforms," *Ada User Journal*, vol. 37, pp. 59–63, Mar. 2016.

[46] R. Giorgi, "AXIOM: A 64-bit reconfigurable hardware/software platform for scalable embedded computing," in *6th Mediterranean Conf. on Embedded Computing (MECO)*, pp. 113–116, June 2017.

[47] R. Giorgi, "Scalable embedded systems: Towards the convergence of high-performance and embedded computing," in *Proc. 13th IEEE/IFIP Int.l Conf. on Embedded and Ubiquitous Computing (2015)*, pp. 148–153, Oct. 2015.

[48] D. Theodoropoulos, S. Mazumdar, E. Ayguade, N. Bettin, J. Bueno, *et al.*, "The axiom platform for next-generation cyber physical systems," *ELSEVIER Microprocessors and Microsystems*, pp. 540–555, 2017.

[49] C. Alvarez, E. Ayguade, *et al.*, "The AXIOM software layers," *ELSEVIER Microprocessors and Microsystems*, vol. 47, Part B, pp. 262–277, 2016.

[50] M. Solinas, R. Badia, *et al.*, "The teraflux project: Exploiting the dataflow paradigm in next generation teradevices," in *IEEE Proc. 16th EUROMICRO-DSD*, (Santander, Spain), pp. 272–279, 2013.

[51] A. Mondelli, N. Ho, A. Scionti, M. Solinas, A. Portero, and R. Giorgi, "Dataflow support in x86-64 multicore architectures through small hardware extensions," in *IEEE Proc. DSD*, pp. 526–529, August 2015.