An Analysis on Virtualization Techniques in Multicore Systems

Abuelgasim Ibrahim Musa, Munam Ali Shah, Hasan M H Owda, Rasool Bakhsh Jatoi
Department of Computer Science
COMSATS Institute of Information Technology
Islamabad, Pakistan
gasim1_78{at}yahoo.com

Abstract— **Since the inception of virtualization technology in recent years, multi-core systems have become the major platform for most of real-world virtualized systems. However, the invention of multi-core CPUs adds complication to the view. This paper provides a critical analysis of the existing virtualization techniques for multi-core systems and outlines its advantages, challenges and the outcome of the application of those techniques on each multi-core platform.**

Keywords-component; Hypervisor, Multi-cores, Virtualization, Virtual Layer.

I. INTRODUCTION

Virtualization concept allows the sharing of hardware (CPUs, memory, I/O) on a single computer, this has come in the sense that multiple operating systems (OSs) to function by sharing the same system. The remarkable advantage of sharing the same system by implementation of Virtualization is the minimization of cost and power efficiency in addition to the flexibility. Virtualization is made by establishing a software layer known as Hypervisor or a Virtual Machine Monitor (VMM)

System Virtualization is being used at an increasing rate in multi-core systems for mainly the benefit of processor consolidation. The sharply increasing cost of a processor core, makes some thoughts that the use of Hypervisors in embedded systems is not a permanent event, and hence will become obsolete as multi-core technology becomes the required standard. In fact it was also noted that, multi-core chips will be always dependent on the efficiency of Hypervisors in order to have efficient resource management.

Our world's virtual solutions consist of four basic components, these are: Tesimulated environment, a client viewer at end user level, a gathering of concerted resources obtainable from inside the virtual world environs, and a network organization that compresses and supports the virtual world solution.

A virtualized setup lets users to convert hardware res A virtualized setup lets users to convert hardware resources into a further elastic software-based resource and more explicitly, arrange for the ability to compile and then reallocate multiple hardware resources like CPUs, memory, storage, and network controllers to generate one or extra completely functional virtual machine. These virtual machines (VM's) can care of their own operating system and applications – thus replicating the same competences of one or more singular physical computing platforms [1, 2].

The additional CPUs you have existing in a PC that runs as a virtualized machine, the additional processing supremacy you can share amongst the virtual PCs. Nevertheless the existence of multi-core CPUs obscures the picture a slightly. Can a PC using four physical CPUs function on a virtualized burden as same as a two-CPU system by means of two cores each CPU? Besides if so, how to sort out anything distinct necessity to be prepared?

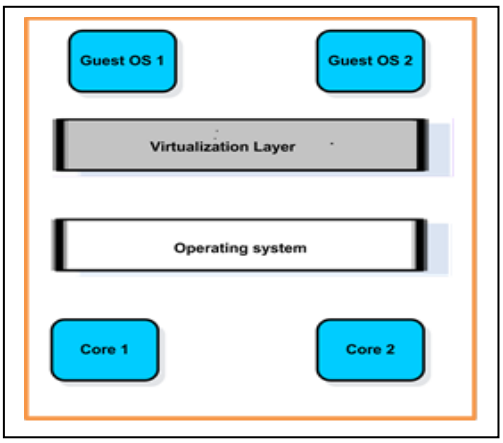

Figure 1: Multi-core Virtualization layers

Multi-core systems took the vantage of Moore's Law by means of putting in extra processors in a particular field. Current commodity multi-core technologies have system Virtualization architectures that offer microprocessor assignment segregation. The sum of CPU-cores, in association to I/O interfaces, is high in the multi-core servers. This results in the sharing of I/O devices among independent virtual machines and therefore, changes the I/O device sharing dynamics, whereas in relationship to, dedicated, non-virtualized servers [3, 4 and 5].

There remain two recognized drives for Virtualization in server environments. First of all is the desire to make a single computer doing as multiple computers to a group individuals .The other main pusher is the starvation to shape legacy code on new PCs. In Multi-core Virtualization , you have a single computer with the Operating System that hosts multiple guest operating system on the same hardware over a Virtualization layer that makes the translations in order to satisfy the of all(Fig 1).

At a more modest level, Virtualization performs the responsibilities of: allocation and sharing of resources for the guest programs. This diagram in figure 1 can be scaled to include more Cores and more guest operating systems, this will add to it more complexity that need to be seen.

Virtualization in such a complex embedded system must offer all of the benefits of allocating, sharing, and isolation required for a server. But, in many cases, the multiple "programs" actually work together on the implementation of a higher-level system, and so they may need to communicate in a way that would not be necessary – or even desirable – between, say, different tenants in a cloud-computing server. So here the Virtualization services would need to be able both to isolate for security and manage communication for the mission system.

A virtual CPU is an abstraction of a hardware CPU that models its behavior. However, a virtual CPU can be equally allocated to any of the existing cores. [4, 7].

A. *Virtualization Techniques*

Virtualization schemes are not same. Some methods may be used differently in implementing subsystems and provide features that other methods do not provide. Primary implementations of Virtualization might be as follows [8].

- System Emulation
- The Native Virtualization
- Para –Virtualization technique
- Virtualization at Operating System Level
- Resources level Virtualization
- Storage Virtualization
- Application level Virtualization

a. *Emulation (EM)*

IS a Virtualization technique in which the entire hardware architecture might be created in a software, this motioned software can replicate the functions of the hardware processor and the associated hardware system .This technique possesses a wonderful flexibility in the sense that the guest OS might not need to be altered to run on. Emulation has features tremendous disadvantages in performance penalties because each instruction on the guest Operating System must be translated prior to be understood by the host system [8, 10].

b. *Native Virtualization (NV)*

It is another technique for providing virtualized guests on a host system. In This method any guest software must have a compatible with the host. It introduces software called a 'Hypervisor', which acts as a command translator between the guest OS and the host hardware. The mentioned Hypervisor may serve several guest systems on a single host, and is found in several Virtualization methods. NV is considered as a middle ground between full emulation, and Para-Virtualization. It does not require any modification in the guest OS to enhance Virtualization capabilities. [8, 11].

c. *Para- Virtualization (PV)*

Abundant systems have been established that use the above mentioned techniques: particular architectures for running virtual machines in, or else fully emulating a system environment. These systems addressed to have the disadvantage of requiring a specialized hardware, compromise fewer than needed recital, or else cannot support the commodity OS [16].

d. *Operating System Level Virtualization (OSLV)*

Is the technique in which an operating system kernel offers multiple individual user-space instances? It's not a true Virtualization; however it does enable user-space applications to run in isolation from different software [8, 12].

e. *Resources Level Virtualization (RV)*

Virtualizing system particular resources similar to storage capacities, namespaces then the linkage resources are identified as resource Virtualization. Around numerous methods to achieve resource Virtualization. Selected of them are [16, 17].

- Combining many separate elements into the bigger resource puddle.
- Grid computing or computer clusters where manifold unconnected computers are pooled to form a big supercomputer by means of huge resources.
- Subdividing a single source such as disk space into a number of reduced and with no trouble reachable resources of the similar type.

f. *Virtual Storage (VS)*

VS are some specific kind of Resource Virtualization in which merits its own subcategory. VS give us a single logical disk from many different systems across a network. The disk is then could be made available to Host or Guest OS's. Storage Virtualization is a practice of Resource Virtualization, where a logical stowage is shaped by conceptualizing altogether the bodily storage resources that are distributed above the network [6, 13]. First the physical storage means are pooled to form a storage puddle which formerly creates the logical storage. This logical storage which is the collection of dispersed physical means looks to be a single huge storage device to the customer [14, 16].

g. *Application Virtualization* (AV)

AV Provides small size single application virtual machines that allow for emulation of a particular environment on a client system. For example a Java Virtual Machine .This Virtualization is limited in the sense that it only provides single

program isolation from the host. It is nevertheless useful when testing programs. [8, 27].

h. MULTI-CORE COMPUTER

Also acknowledged as a chip multiprocessor syndicates two or extra Processors named (Cores) on a single piece of silicon (termed a die). Naturally, each core contains the entirely elements of an independent CPU, for example registers, ALU, pipeline hardware, then control unit, and above L1 instruction and data caches. In addition to the several cores, modern multi-core chips likewise include L2 cache and, in certain cases, L3 cache. The possible performance remunerations of a multi-core organization rest on the capability to efficiently exploit the parallel resources existing on the way to the application. [9].

II. RELATED WORK

Virtualization concept was initially presented by IBM in the 1960s to offer synchronized, interactive access to a mainframe computer - IBM 360, which cares many instances of operating systems running on the same hardware platform Normally, VMs are alienated into two major types: process VMs and system VMs. [10] some authors deliberated system VMs whereas others talk over the process virtual machine. The Virtual Machine Monitor (VMM) is a essential constituent of system VM, that delivers the anticipated abstraction layer of the hardware for each operating system (OS) running on it, for example VMware, Virtual PC, and Xen [11,30].

A. The Initiation of Virtualization

A HAL level software was implemented, called a virtual machine monitor (VMM), lying between raw hardware and virtual machines (VMs), to give the guest OSs a virtualized view of all the hardware. VMM manages all the VMs where every VM provides facilities to an OS or application to believe as if it runs in a normal environment and directly on the same hardware. Companies like Intel and AMD presented their CPU products with Virtualization support since 2005 and 2006, respectively. With the support of hardware, developers can build a much tidier VMM. Virtualization functions at the OS level work on top of or as a module in OS to provide a virtualized system call interface. In this kind of virtual environment, the kernel of an operating system allows for multiple isolated user-space instances (instead of just one instance). These instances (often named containers, VEs or VPSs) look like real servers, from the standpoint of their owners. Because of this, OS level Virtualization is also named single OS image Virtualization or container based Virtualization. It usually imposes little or no overhead [11, 30].

B. The VMM

VMM is usually a small OS nonetheless with no hardware drivers. For getting into the physical resources, the VMM is naturally attached with a normal operating system, like Linux, which offers device/hardware access. There are two methods employed, formalized by Goldberg as: (a) type-I Virtualization wherever the VMM and VM run straight on the physical hardware, then (b) type-II Virtualization where the VMM and VM run on a host operating system. Since the type-I Virtualization has direct access to resources; performance is comparable to that of native execution. In contrast, type-II Virtualization incurs the cost of additional overhead due to the layering of the VMM on top of the host OS when servicing resource requests from VMs. The type-II layering makes its approach more suitable for the development phase, where some performance may be reduced in exchange for greater diagnostic and development capabilities. Today, several system-level Virtualization solutions are available, for instance Xen (type-I), QEMU (type-II), or VMware workstation & server (type-II). However, these resolutions are not suited for high performance computing (HPC). Xen has become a rather massive micro-kernel that includes unneeded features for HPC, e.g., a network communication bus; QEMU and VMware do not offer support to direct access to high-performance network solutions [12, 29, and 22].

C. The Challenges of Multi-core

It is difficult to overestimate the magnitude of the discontinuity that the high performance computing (HPC) community is about to experience because of the emergence of the next generation of multi-core and heterogeneous processor designs [5, 22, and 28]. For at least two decades, HPC programmers have taken it for granted that each successive generation of microprocessors would, either immediately or after minor adjustments, make their old software run substantially faster[21]. But three main factors are converging to bring this “free ride” to an end. First, system builders have encountered intractable physical barriers – too much heat, too much power consumption, and too much leaking voltage – to further increases in clock speeds. Second, physical limits on the number of pins and bandwidth on a single chip means that the gap between CPU performance and memory performance, which was already bad, will get increasingly worse.

Ultimately, the design trade-offs being made to address the previous two factors will render commodity processors, absent any further augmentation, inadequate for the purposes of tetra- and peta-scale systems for advanced applications[25]. This daunting combination of obstacles has forced the designers of new multi-core and hybrid systems, searching for more computing power, to explore architectures that software built on the old model are unable to effectively exploit without radical change. But despite the rapidly approaching obsolescence of familiar programming paradigms, there is currently no well understood alternative in whose viability the community can be confident [20, 23]. The core of the problem is the dramatic increase in complexity that software developers will have to face. Dual-core machines are already common, and the number of cores is expected to roughly double with each processor generation. But contrary to the premises of the previous model, programmers will not be able to consider these cores independently (i.e. multi-core is not “the new SMP”) because they share on-chip resources in ways that separate processors do not. This position is made even more

complicated by the other non-standard components that future architectures are expected to deploy, including mixing different types of cores, hardware accelerators, and storage systems. Finally, the proliferation of widely divergent design ideas shows that the question of how to best combine all these new resources and components is largely unsettled. When combine changes produce an impression of a future in which software engineers must overcome software design problems that are vastly more complex and challenging than in the past in order to take advantage of the much higher degrees of concurrency and greater computing power that new architectures will provide. [13, 26].

D. Main factors driving the multi-core discontinuity

Among the various factors that are driving the momentous changes now occurring in the design of microprocessors and high end systems, three stand out as especially notable: 1) the number of transistors on the chip will continue to double roughly every 18 months, but the speed of processor clocks will not continue to increase; 2) the number of pins and bandwidth on CPUs are reaching their limits and 3) there will be a strong drift toward hybrid systems for peta-scale (and larger) systems. The first two involve 2 fundamental physical limitations that nothing currently on the horizon is likely to surmount. The third is a consequence of the first two, combined with the economic necessity of using many thousands of CPUs to scale up to peta-scale and larger systems.

Each of these factors has a somewhat different effect on the design space for future programming: 1) More transistors and slower clocks means multi-core designs and more parallelism required – The modus operandi of traditional processor design – increase the transistor density, speed up the clock rate, raise the voltage – has now been blocked by a stubborn set of physical barriers – too much heat produced, too much power consumed, too much voltage leaked. Multi-core designs are a natural response to this post. By putting multiple processor cores on a single die, architects can continue to increase the number of gates on the chip without increasing the power densities. But since excess heat production means that frequencies cannot be further increased, deep-and-narrow pipeline models will tend to recede as shallow-and-wide pipeline designs become the norm. Moreover, despite obvious similarities, multi-core processors are not equivalent to multiple-CPUs or to SMPs. Multiple cores on the same chip can share various caches (including TLB!) and they certainly share the bus. Extracting performance from this configuration of resources means that programmers must exploit increased thread-level parallelism (TLP) and efficient mechanisms for inter-processor communication and synchronization to manage resources effectively. The complexity of parallel processing will no longer be hidden in hardware by a combination of increased instruction level parallelism (ILP) and deep-and-narrow pipeline techniques, as it was with superscalar designs. It drive have be addressed in software [13, 27]. 2) Thicker "memory wall" means that communication efficiency will be even more essential – The pins that connect the processor to main memory have become a strangle point, with both the rate of pin growth and the bandwidth per pin slowing down, if not flattening out. Thus the processor to memory performance gap, which is already approaching a thousand cycles, is expected to grow, by 50% per year according to some estimates. Concurrently, the number of cores on a single chip is expected to continue to double every 18 months, and since limitations on space will keep the cache resources from growing as quickly, cache per core ratio will continue to decline. Problems of memory bandwidth, memory latency, and cache fragmentation will, therefore, tend to get worse. [13, 14 and 15]

3) Limitations of commodity processors will further increase heterogeneity and system complexity: Experience has shown that tera- and peta-scale systems must, for the sake of economic viability, use commodity off-the-shelf (COTS) processors as their foundation. Regrettably, the trade-offs that are being (and will continue to be) made in the architecture of these general purpose multi-core processors are unlikely to deliver the capabilities that leading edge research applications require, even if the package is suitably qualified. Therefore, in addition to all the different kinds of multithreading that multi-core systems may utilize – at the core-level, socket-level, board-level, and distributed memory level – they are also likely to incorporate some constellation of special purpose processing elements. Examples include hardware accelerators, GPUs; off-load engines (TOEs), FPGAs, and communication processors (NIC-processing, RDMA). Since the competing designs (and design lines) that vendors are offering are already diverging, and mixed hardware configurations are already appearing, the hope of finding common target architecture around which to develop future programming models seems at this point to be largely forlorn. It is believed that these major trends will define, in large part at least, the design space for scientific software in the coming decade. But while it may be important for planning purposes to describe them in the abstract, to appreciate what they mean in practice, and therefore what their strategic significance may be for the development of new programming models, one has to look at how their effects play out in[13, 24].

E. Platform Feature Comparison

With the wide array of potential choices of virtualization technologies available, it's often difficult for potential users to identify which platform is best suited for their needs. In order to simplify this task, some authors conducted a comparison between Xen 3.1, KVM from RHEL5, VirtualBox 3.2 and VMware ESX. The first point of investigation is the Virtualization method of each VM. Each Hypervisor supports full Virtualization, which is now common practice within most x86 Virtualization deployments today. Xen, originating as a para-virtualized VMM, still supports both types, however full Virtualization is often preferred as it does not require the manipulation of the guest kernel in any way. From the Host and Guest CPU lists, we see that x86 and, more specifically, x86-64/amd64 guests are all universally supported. Xen and KVM both support Itanium-64 architectures for full Virtualization (due to both Hypervisors dependency on QEMU), and KVM also claims support for some recent PowerPC architectures. VirtualBox and VMware have internal mechanisms to provide full Virtualization even without the Virtualization instruction sets, and Xen can default back to Para-virtualized guests. Considering host environments for

each system , As Linux is the primary OS type of choice within HPC deployments, its key that all Hypervisors support Linux as a guest OS, and also as a host OS. As VMware ESX is meant to be a Virtualization-only platform, it is built upon a specially configured Linux/UNIX proprietary OS specific to its needs. All other Hypervisors support Linux as a host OS, with VirtualBox is also supporting Windows. VirtualBox, on the other hand, supports only 32 vCPUs and 16GB of addressable RAM per guest OS, which may lead to problems when looking to deploy it on large multi-core systems.. Another vital aspect of these Virtualization technologies is the license agreements for its applicability within HPC deployments. Xen, KVM, and VirtualBox are provided free of charge under the GNU Public License (GPL) version 2. VMware, on the other hand, is completely proprietary with an extremely limited licensing scheme that even prevents the authors from will fully publishing any performance benchmark data without specific and prior approval . [14, 18 and 19].

III. Evaluation of implementing Virtualization in Multicore Systems

Different Virtualization techniques and technologies in Multicores platforms were reviewed and critically analyzed .The previous discussed techniques are applicable with some advantages and limitations when applied in different Virtual Machine Mangers (VMM), though the implementation of some showed their usefulness in certain real life problems . Recent studies have shown the industry direction to support Multicore systems to better utilization of their hardware capabilities in the form of software designs that has ability to run better in these cores.

The choice of the right hypervisor was the sign of success for some implementation. Previous studies also showed that certain Virtualization techniques in some multicore platform can even be useful for future designs .Virtualization Techniques implementation was found to highly dependent on platform, hypervisor used and the Operating system.

Table 1 shows summaries on each Virtualization technique and the applicability in cases of certain platforms and hypervisors. The table also shows the advantages and limitations addressed by previous studies.

Techniques	**Results From previous Implementation**				
	Hypervisor	Platform	Advantages	Applicability	Limitations
Para-Virtualization	Xen	HPC	Easy to grow and widely used as a flavoring	Not recommended for HPC	The performance lacks
Para-Virtualization	RT-Xen	Dell Q9400 Quad core	Not addressed	Promising step to real-time Virtualization	Not addressed in statutes reviewed
Full-Virtualization	DBT	ARM V7-A	DBT better than hardware assisted Virtualization and has possibility of development	Applicable in modern Computer architectures which not classically vertualizable	Implementation cost and complexity and memory footprint
Para-Virtualization	Xen (RT-Xen2.0)	Inteli7 x980 with 6 Cores	Lower deadline misses compared to other real-time schedulers	Can be implemented in dynamic memory management	It takes time longer than theoretical predicted time
Full-Virtualization	KVM with EPT and PLE support	Dell Power Edge R720 Server with 2 2.4GHZ Intel Xeon ES-2665 CPU 8 cores	Reduction in slow down and actively Cores	If well studied , Can be a primary solution for synchronization I virtualized environments	Not well studied and alternative designs are needed
Para-Virtualization	Xen4.1.1	Quad core Xeon Y5560 8GB DDR memory	Dynamically adjusting allocations to meet changes in workload	Application level quality of service not addressed	Applicable in power allocation over provision multicore platforms
Para-Virtualization	MultiPARTES XtratuM	Dual core x86 processor and FPGA with LEON3 (spare V8) synchronized processors	Has a methodology to deal with different levels of criticality and layers	Issues related to OS used and security	Able to adapt the MultiPARTES
Full-Virtualization	KVM	Quad Core physical machine	Performance enhancement for optimal processor scheduling	Not verified to optimize big data tools to utilize multicore architectures	Can be used to quantify the overhead from CPU migrations for heterogeneous workloads in systems
Para-Virtualization	Xen4.2.1	Two 2.53GHZ Intel Xeon E5540 processors 4 cores each and 8 MB L3 Cache	Added a time slice and result in better performance in computation tasks	Xen default credit scheduler has problems with the I/O performance of mixed workloads	Applicable in new multicore based schedulers with better performance.
OS level Virtualization	---------- -	Current X86 processors	Better performance than Xen	Dynamic resource reallocation not supported	Suitable for real-time workloads that have fixed resource demand
Para-Virtualization	Xtratum	LEON4	LEON$ Overcomes some bottlenecks by implementing 128-bit bus	Still there is a memory access bottleneck And requirement of changes	Can support SMP hardware architecture
Full system emulation	QEMU	X86	Ability to simulate I/O	Significant complexity handling in virtual memory issues	Useful in modelling heterogeneous core designs of the future
Para-Virtualization	Xen3.2.1	Intel quad Core Xeon with each RAM and two 1GB NIC card	Well-matched for enterprise IP telephony	Media applications are challenging direct I/O access is essential from time to time	Can be employed in telecommunication solution providers

TABLE 1 ADVANTAGES AND LIMITATIONS

REFERENCES

[1] Lesko, Charles J., and Yolanda A. Hollingsworth. "Architecting Scalable Academic Virtual World Grids: A Case Utilizing OpenSimulator." Journal of Virtual Worlds Research 6.1 (2013). W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[2] Olukotun, Oyekunle Ayinde. Multicore processors and systems. Eds. Stephen W. Keckler, and H. Peter Hofstee. Springer, 2009.

[3] Reilly, Matthew. "When multicore isn't enough: Trends and the future for multi-multicore systems." Proceedings of Workshop on High Performance Embedded Computing. 2008.

[4] B. Smith, "An a Carrascosa, E., et al. "XtratuM hypervisor redesign for LEON4 multicore processor."pproach to graphs of linear forms (Unpublished work style)," unpublished.

[5] Petrides, Panayiotis, et al. "Virtualization for morphable multi-cores." ARCS 2011 (2011).

[6] Bini, Enrico, et al. "Resource management on multicore systems: The ACTORS approach." Micro, IEEE 31.3 (2011): 72-81.

[7] Reilly, Matthew. "When multicore isn't enough: Trends and the future for multi-multicore systems." Proceedings of Workshop on High Performance Embedded Computing. 2008.

[8] White, Joshua, and Adam Pilbeam. "A survey of virtualization technologies with performance testing." arXiv preprint arXiv:1010.3233 (2010).

[9] Stallings, William. Computer organization and architecture: designing for performance. Pearson Education India, 1993.

[10] Shimada, Hiromasa, et al. "Design issues in composition kernels for highly functional embedded systems." Proceedings of the 2011 ACM Symposium on Applied Computing. ACM, 2011

[11] Jin, Hai, et al. "ChinaV: Building virtualized computing system." High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on. IEEE, 2008

[12] Vallee, Geoffroy, et al. "System-level Virtualization for high performance computing." Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on. IEEE, 2008.

[13] Dongarra, Jack, et al. "The impact of multicore on computational science software." CTWatch Quarterly 3.1 (2007): 1-10

[14] Younge, Andrew J., et al. "Analysis of virtualization technologies for high performance computing environments." Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011

[15] Reilly, Matthew. "When multicore isn't enough: Trends and the future for multi-multicore systems." Proceedings of Workshop on High Performance Embedded Computing. 2008

[16] Rose, Robert. "Survey of system virtualization techniques." Collections (2004).

[17] Sahoo, Jyotiprakash, Subasish Mohapatra, and Radha Lath. "Virtualization: A survey on concepts, taxonomy and associated security issues." Computer and Network Technology (ICCNT), 2010 Second International Conference on. Ieee, 2010.

[18] Ding, Xiaoning, Phillip B. Gibbons, and Michael A. Kozuch. "A Hidden Cost of Virtualization when Scaling Multicore Applications."

[19] Penneman, Niels, et al. "Formal virtualization requirements for the ARM architecture." Journal of Systems Architecture (2013).

[20] Trujillo, Salvador, Alfons Crespo, and Alejandro Alonso. "MultiPARTES: Multicore virtualization for Mixed-criticality Systems." Digital System Design (DSD), 2013 Euromicro Conference on. IEEE, 20

[21] Velkoski, Goran, Sasko Ristov, and Marjan Gusev. "Affinity-aware HPC applications in multichip and multicore multiprocessor." Information Technology Interfaces (ITI), Proceedings of the ITI 2013 35th International Conference on. IEEE, 2013.

[22] Lim, Seung-Hwan, et al. "Performance Implications from Sizing a VM on Multi-core Systems: A Data Analytic Application's View." Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International. IEEE, 2013

[23] Wen, Yuanfeng, et al. "Multiprocessor architectural support for protecting virtual machine privacy in the interested cloud environment." Proceedings of the ACM International Conference on Computing Frontiers. ACM, 2013.

[24] Yu, Chao, Leihua Qin, and Jingli Zhou. "A multicore periodical preemption virtual machine scheduling scheme to improve the performance of computational tasks." The Journal of Supercomputing (2013): 1-23.

[25] Dai, Yuehua, et al. "Design and verification of a lightweight reliable virtual machine monitor for a many-core architecture." Frontiers of Computer Science(2013): 1-10

[26] Herber, Christian, et al. "Self-virtualized CAN controller for multi-core processors in real-time applications." Architecture of Computing Systems–ARCS 2013. Springer Berlin Heidelberg, 2013. 244-255.

[27] Cerotti, Davide, et al. "End-to-End Performance of Multi-core Systems in Cloud Environments." Computer Performance Engineering. Springer Berlin Heidelberg, 2013. 221-235.

[28] Smari, Waleed W., Sandro Fiore, and David Hill. "High performance computing and simulation: architectures, systems, algorithms, technologies, services, and applications." Concurrency and Computation: Practice and Experience (2013).

[29] Gupta, Vishal, and Karsten Schwan. "PowerTune: Differentiated Power Allocation in Over-provisioned Multicore Systems."

[30] Shih, Chi-Sheng, et al. "Fairness scheduler for virtual machines on heterogonous multi-core platforms." ACM SIGAPP Applied Computing Review13.1 (2013): 28-40

.