Making the Case for Highly Efficient Multicore Enabled Unikernels With IncludeOS

Conference Paper · February 2017

CITATION
1

READS
434

3 authors, including:

Maghsoud Morshedi Chinibolagh
University of Oslo
8 PUBLICATIONS **18** CITATIONS
SEE PROFILE

Hårek Haugerud
Oslo Metropolitan University
52 PUBLICATIONS **544** CITATIONS
SEE PROFILE

Making the Case for Highly Efficient Multicore Enabled Unikernels With IncludeOS

Maghsoud Morshedi, Hårek Haugerud, Kyrre Begnum

Dept. of Computer Science
Oslo and Akershus University College of Applied Sciences
Oslo, Norway
Email: { maghsoud.morshedi|haugerud|kyrre.begnum } @hioa.no

***Abstract*—Today's data centers utilized for cloud services represent a significant energy consumption and costs. Standard operating systems used for cloud instances are still designed largely to run on actual or emulated hardware, making them wasteful when being idle. Ideally, the cloud should be populated with leaner and more efficient operating systems. Unikernel operating systems are a good example of such, but most Unikernels are still not ready to be used in a cloud as they are built on specialized emulators. Furthermore, they are designed for single core operation and it is impractical to run hundreds or thousands of virtual machines for large workloads without straining the underlying cloud platform. The idea presented in this paper is to have all benefits of a lean Unikernel operating system while equipping it with multicore capabilities in order to represent an energy efficient and cloud-optimized operating system that can handle larger computations. IncludeOS has shown to be an extremely efficient Unikernel operating system, utilizing a much simpler event handler and foregoing the timer interrupt altogether. In our case, the experiments demonstrated increased performance for a multi-threaded processor intensive task compared to a classic operating system, thus showcasing a real-life solution for energy efficient computation in cloud environments.**

Keywords–Cloud computing; energy efficiency; green computing; Unikernel; multicore computing.

I. INTRODUCTION

Cloud computing has been focused on offering cost reduction for the consumer, business and scientific domains. However, significant energy consumption of data centers has started to constrain scaling and further cost reduction because of electricity bills and carbon dioxide footprints.

Numerous dedicated approaches for energy efficiency in cloud environments have been proposed. One traditional approach in optimising energy efficiency in such environments, is through operating system (OS) virtualization, which allows for multiple virtual machine (VM) to run on a shared cluster of physical machines. In this context, a VM represents a complete computer system with a standard OS and typically a host of a single application. The VMs can be consolidated and relocated in order to reduce energy waste. In this line of thinking, however, little attention has been paid to the role of the operating system.

By design, standard operating systems are multipurpose and are intended to run on hardware with a variety of device drivers. This allows them to support a diversity of services on physical and emulated hardware with little modification, but makes them wasteful in times when they are idle. One clear example is the timer interrupt, which triggers the kernel of an operating system to wake up at a regular pace to look for device activity. In a virtual machine, where there are very few "hardware" devices, the kernel still emulates that behavior, resulting in scores of VMs waking up and spending CPU cycles thousands of times every second. As a result, todays general purpose operating systems, though convenient, constitute a continuous energy leak for todays data centers and cloud environments.

In addition, there are also challenges which arise due to processor design. Processor architecture has evolved from featuring a single high-frequency processor, to having multiple low-frequency processor cores. This development was partly driven by frequency increment constraint on a single processor - better known as the frequency wall[1].

In contrast to a standard operating system, a Unikernel operating system is designed for a single purpose - where a single service is bundled with only the essential libraries [2], and is not designed to run on hardware. Unikernel operating systems are capable of delivering optimal performance as well as low resource consumption. However, cloud systems have not been adapted to support Unikernel operating systems due to the specialized nature of the required emulators.

Multicore processors have become the dominant processor type, and have experienced a continuous growth in the number of cores on a single processor, over time. The design of currently available Unikernels does not take advantage of the presence of multiple cores, as their operations are bound for execution on a single core. This contributes to a diminishing performance as the number of Unikernel virtual machines is gradually increased on a single host. The deployment of a large federation of single-core Unikernel VMs is impractical for a sizable workload as it strains the underlying cloud layer.

On-going Unikernel development projects are at different stages of maturity. Prominent among them is IncludeOS, which is under development at the Oslo and Akershus University College of Applied Sciences. It is being developed primarily in C++, to run on the quick emulator (QEMU)/kernel-based virtual machine (KVM) hypervisor, but with the potential of being ported to other platforms with slight modification to its binary. IncludeOS is an efficient Unikernel operating system, which utilizes a simple event handler with little memory overhead: when running a domain name system (DNS) service it imposes a total memory footprint of 158KB [3]. IncludeOS uses no regular timer interrupt, meaning that at idle, the virtual machine will use no central processing unit (CPU) cycles. Although it is efficient, IncludeOS has been a single-core operating system and not been able to utilize multiple cores for scientific and CPU-bound workloads.

 ISBN: 978-1-61208-529-6

This paper presents our approach to equip the IncludeOS Unikernel with multicore capabilities so that it can handle large workloads efficiently. By using multicore computing, a Unikernel operating system can handle a large processor intensive computation concurrently so that it enhances performance. The rest of the paper is organized as follows:

- The existing IncludeOS architecture and limitation along with typical challenges of multicore computing appear in Section II. In addition, this section proposes possible applications for multicore Unikernels.
- The design principles that form the multicore Unikernel architecture appear in Section III. We identify race conditions and utilize an efficient technique in Section III-A in order to minimize energy waste. Handling and distributing tasks in a multicore system appears in Section III-B. Section III-C presents our intercommunication scheme among the logical processors.
- The developed multicore capability for IncludeOS Unikernel operating system is evaluated compared to multiple single-core IncludeOS, Ubuntu VM and bare metal Ubuntu. Hence, Section IV presents the results of our experiments while Section V evaluates them.
- Section VI presents related projects in the scope of multicore Unikernel development followed by conclusion and future work in Section VII

II. Common Challenges

A standard operating system was initially designed for a single-core processor. The transition to multicore hardware technology is a slow process due to the incredible complexity of today's established operating system kernels. For example, many of the algorithms used in a standard operating system cannot take advantage of complete multicore capabilities while cores are in full power state. Hence, multicore computing can not guarantee a sufficient performance to energy ratio improvement despite an increase in clock speed.

The adoption of multicore computing poses critical challenges in software development, which influence energy efficiency. An operating system with full parallelism will utilize all of the available computing power of a multicore processor. On the other hand, an operating system with little or no parallelism will consume more energy in comparison to their output in a multicore system. Therefore, operating system must manage cores so that each core can execute independent instruction streams concurrently in order to maximize energy efficiency for a large workload.

Multicore processors require a new generation of operating systems that capitalize form the available computing power with low energy consumption. Hence, Unikernels as a new generation of operating systems must address the fundamental challenges presented by multicore computing.

A. Multicore Unikernel Applications

There are many compute-intensive applications in science, research and engineering that demand parallelism. With a minimal footprint and multicore computing, Unikernels could enable scientist, researchers and engineers to deploy their solutions in an efficient way. Researchers in the field of bioinformatics analyse new sequences of DNA or protein in order to predict their biological function. There are couple of packages that utilize profile-hidden Markov models (HMM) in order to search for and align sequences. These packages are very processor-intensive and utilize more than 99 percent of a single-core processor while they generally have the capacity of being parallel[4].

The telecommunication industry is recognising the possibility of cloud software defined radio (SDR) as an evolving technology. The SDR perquisites of processor-intensive digital signal processing, real-time throughput and minimum latency, show the potential of multicore Unikernel as a SDR node.

Unikernels can be leveraged as simple caching and in-memory storage solutions. In todays data-driven infrastructures, efficient, distributed databases can be built using Unikernel operating systems.

B. IncludeOS design

The IncludeOS Unikernel operating system was designed with a modular architecture in mind such that it enables developers to attach their C++ service code to the operating system kernel during compile time, which eliminates the overhead of system calls. This provides IncludeOS the capability of attaching just what a service actually needs and minimizes the memory footprint by excluding unused features.

Application developers will write their service applications as a normal C++, standard library application. However, when including the IncludeOS library in their code with the simple addition of `#include <os>`, and subsequently compiling the code using the IncludeOS toolchain, the end result is not just a binary of the application, but a standalone, bootable virtual machine image where the operating system components that are needed by the application are statically linked into the file. This image can then be booted using QEMU/KVM and is compatible with popular cloud environments like OpenStack.

The IncludeOS comprises a modular network stack connected to the only VirtioNet device driver so that it reduces the overhead of other protocols for a service which does not use them. For example, if the application only uses TCP sockets, no UDP support will be added during compile time. Beside the modular network stack, IncludeOS' asynchronous I/O setup uses a counter based approach in order to eliminate context switching during the interrupt handling. The IncludeOS memory footprint is quite small, which enables IncludeOS to boot up quickly in about 0.3 seconds. All of the design considerations enable IncludeOS to be a lean single-threaded operating system, which can handle one task at a time.[3]. Likewise, Bratterud et al.(2015) presented detailed architecture of IncludeOS.

III. Design Principles

The following part presents our design principles in order to adapt IncludeOS to support multicore computing. In a nutshell, multicore IncludeOS will utilize a design of a master processor managing multiple application processors. The developer writing an IncludeOS based application will organize the parallel workloads as tasks in the code. Once the VM is running, the initial bootstrap processor will become the master and distribute the tasks among the available application processors. The master will also execute task workloads in order to optimize the efficiency.

 ISBN: 978-1-61208-529-6

A. Energy Efficient Memory Access

In a multicore system, parts of a program may be executed concurrently by more than one core so that it requires mutual exclusion of access over a critical memory section [5]. Any multicore operating system should employ mutual exclusion access over critical sections in order to guarantee serialized access.

In multicore IncludeOS, a bootstrap processor is responsible for booting the operating system, which then will wake up application processors in order to take advantage of them. Since application processors have been awakened through a broadcasted inter-processor interrupt (IPI) call, they can run self-configuration code concurrently following the principle of single program, multiple data [6].

The application processors will encounter a critical section problem in the early stage of the initialization procedure. The challenge begins while application processors run the self-configuration code concurrently and will manipulate a common memory location. A common option to handle this situation is to use a semaphore lock in order to serialize concurrent access to a specific memory location. Semaphore locks also introduce a new problem in virtual machines as they cost extra processor cycles to function.

The multicore IncludeOS employed instead a bus locking mechanism while manipulating a critical section in memory in order to prevent electricity waste by using semaphores. The LOCK instruction causes the bus to be locked so that the underlying hardware will manage the race condition and make an instruction atomic. Logical processors connected to the system bus generally use a low priority mechanism in order to deal with race conditions during bus acquisition. In addition, locking the bus simplifies the development of multicore support in an operating system.

B. Multicore Task Management

In a multicore system, the operating system must manage tasks properly in order to maximize performance. There are two main task scheduling mechanisms: preemptive and non-preemptive. Standard operating systems use preemptive task scheduling in order to share limited resources between multiple tasks. Likewise, hypervisors utilize preemptive scheduling while they oversubscribe limited resources to virtual machines. Oversubscription forces hypervisors to do context switching among the available physical resources.

Multicore IncludeOS has followed the idea to keep the preemptive scheduling only at the hypervisor level. Hence, it employs a non-preemptive task management such that it adopts many virtual processors in order to handle a large workload efficiently. Indeed, the hypervisor allocates as many virtual processors as the multicore IncludeOS requires in order to handle large workloads. Fig. 1 illustrates the multicore IncludeOS operating system task management approach in which each task is handled by one core in the multicore IncludeOS.

In addition, the non-preemptive task management provides energy efficiency by reducing memory consumption and avoid context switching. The preemptive scheduling requires a bigger stack size in order to store the state of switched tasks in memory. Hence, an operating system requires more memory for a program stack whenever the number of cores increases. On the other hand, by employing non-preemptive task management

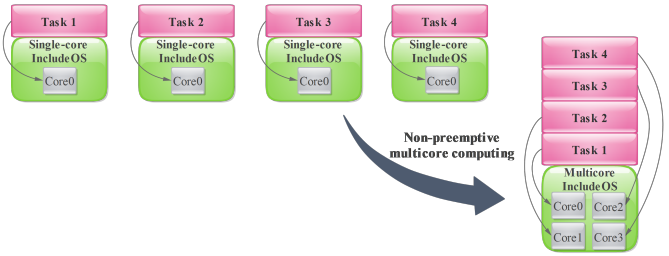


Figure 1. Multicore IncludeOS non-preemptive task management approach.

and avoiding context switching inside the operating system, the operating system can achieve fairness through multicore computing.

Distributing tasks among the virtual processors is another aspect of task management, which affects energy efficiency. The fact is that execution of a task on a logical processor when its sibling is idle is faster than when its sibling is executing a task too. This is due to how hyper-threading technology shares execution resources of each core in order to execute two or more separate threads concurrently[7]. In a processor that supports hyper threading technology, running one task per core enhances performance but at the same time the processor consumes extra electricity. In order to save power, multicore IncludeOS utilized sibling logical processors in one core and wakes the logical processors up whenever they are needed. This enables the hypervisor to change the power state of idle cores to an energy efficient state.

C. Multicore Synchronization

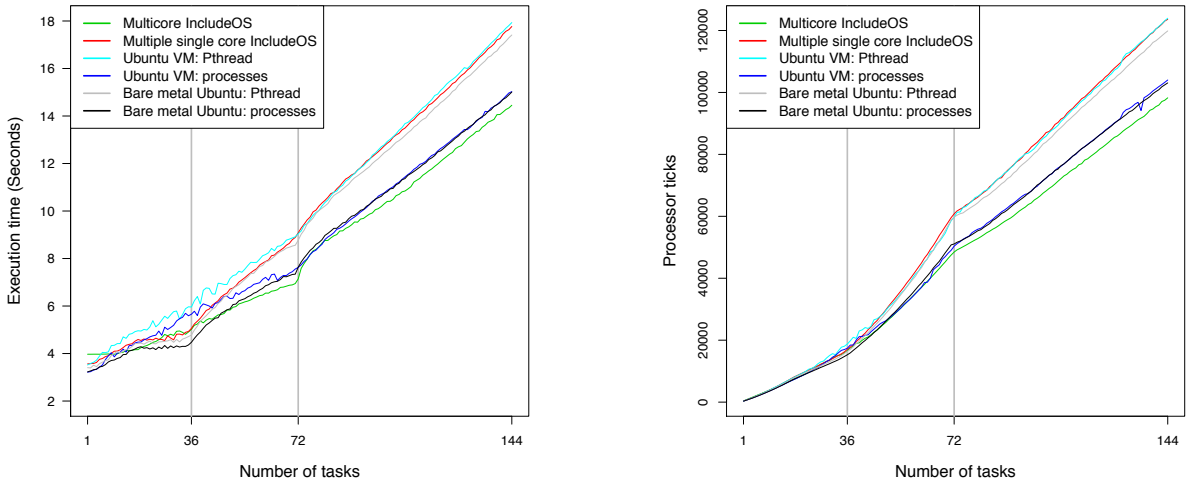
A processor may require communicating with other processors in a system. A bootstrap processor in a multicore system should be able to feed in outputs of logical processors. Shared memory and message passing are the two main techniques for inter-processor communication.

Multicore IncludeOS employs shared memory in order to avoid the complexity and extra overhead of message passing between logical processors. Our design utilized the advanced programmable interrupt controller (APIC) ID in order to build an indexed array of shared memory such that logical processors access their own address space. Since multicore IncludeOS implements a master-slave architecture in order to manage application processors, each application processor plays a producer role and stores its execution result to a particular memory location identified by the APIC ID. The bootstrap processor acts as a consumer and checks the particular location for new data. Indeed, the bootstrap processor may employ busy waiting in order to check whether producers have written data in the agreed memory location. Although busy waiting for a memory location is not an efficient method, multicore IncludeOS utilized the bootstrap processor to execute tasks, as well in order to avoid wasting the processor cycles for busy waiting. In addition, the monitor/mwait mechanism eliminates busy waiting and causes the processor entering a power optimized state while waiting for a change in memory[8]. One should note that hypervisors need to support monitor/mwait before operating systems can utilize it.

IV. Results

In order to assess the performance and efficiency of multicore IncludeOS, we compared the workload performance

 ISBN: 978-1-61208-529-6



(a) (b)

Figure 2. Execution time of prime number computation in multicore IncludeOS, multiple single-core IncludeOS, Ubuntu VM and bare metal Ubuntu with a different number of tasks in the Intel server with 36 cores supporting hyper-threading technology. Fig. (a) illustrates the execution time of workloads in seconds. Fig. (b) illustrates hypervisor processor ticks for each solution.

against a standard Ubuntu virtual machine, multiple single-core IncludeOS instances, as well as a bare metal Ubuntu installation. A series of experiments were conducted with the same processor-intensive binary being executed on all multi-threaded solutions. On the Ubuntu operating systems, parallelism was achieved both through the POSIX thread (Pthread) model and through standard processes scheduled by the kernel. In the case of multicore IncludeOS, executing multiple tasks simultaneously was achieved by making the master processor distribute the tasks to each application processor. For single-core IncludeOS, the parallelism was achieved by running one Unikernel instance for each of the tasks.

Execution time and processor ticks of the virtual machines were measured from the host in order to evaluate efficiency. The task used in the experiments was to calculate the number of prime numbers below a given large number, which is a CPU-bound task. The tasks were distributed by sending the given large number through UDP to the server for calculation. Then the server calculated the largest prime N times using N independent tasks. The sum of theses N numbers was returned through UDP and the time for the whole process recorded. In the special case of single-core IncludeOS, the large number was sent through UDP to N single-core instances, each instance returned a result through UDP and the sum was then calculated.

The experiments were performed on two machines, each with a different processor architecture. Table I shows the specification of both servers. One of the machines was equipped with Intel CPUs, which support hyper-threading technology while the second machine was equipped with AMD CPUs. Our experiments were conducted with an increasing number of task threads/application processors from 1 to twice the amount of available physical CPUs. In the case of the Ubuntu VM, the number of threads or processes was varied while with multiple single-core IncludeOS instances, the number of IncludeOS virtual machines was varied. Each experiment was repeated 30 times.

TABLE I. SERVERS SPECIFICATION.

Platform segment	*Dell server*	*Dell server*
Processors	Intel(R) Xeon(R) CPU E5-2699 v3	AMD Opteron 6234
Processor's frequency	2.3 GHz	2.4 GHz
Memory	128 GB	128 GB
Number of processor sockets	2	4
Number of cores	36	48
Number of logical processors	72	48

Fig. 2a shows the execution time of the prime number calculation workload for six different multi-threading solutions on the Intel server. One might expect the execution time to be roughly independent of number of tasks when there are less tasks than physical cores. The number of cores is here 36, as is indicated by the gray vertical line. However, this is not the case for the Ubuntu VM, the execution time increases by roughly 50% when increasing the number of tasks from 1 to 36. For the other systems, the increase in time is not as profound, but on the other hand it is not flat as would be expected if the system utilized the parallelism of the physical cores perfectly. The experiments below 36 cores are the most important ones as they do not involve overprovisioning of the cores. Except for the case of very few cores, multicore IncludeOS performs better than the Ubuntu VM and equally well as bare metal Ubuntu, which is included as a reference. It also performs better than the single core IncludeOS solution.

When the number of tasks increases from 36 to 72, some of the jobs needs to share an arithmetic logic unit (ALU) as there are only 36 hyper-threading cores, and the execution time is almost doubled. The slope is even steeper from 72 to 144 tasks and a bit larger than 2, which makes sense since then time-sharing is unavoidable. In these regions, the multicore

 ISBN: 978-1-61208-529-6

IncludeOS solution is even more efficient than its Ubuntu counterparts.

It was assumed that using Pthreads would be the most efficient way to run parallel tasks using a Linux OS and that would give the most fair comparison. As Pthreads are known to induce some overhead in certain cases, we also ran the tasks forking ordinary processes and this turned out to be more efficient in our case. This can be seen in Fig. 2a, the process results for the Ubuntu variants outperforms the Pthread results.

Fig. 2b shows the total number of processor ticks performed by the hypervisor during the same experiments, which is a measure of the grand total of CPU resources needed by each of the solutions in order to perform the same calculation. It depicts that multicore IncludeOS consumes a similar amount of processor ticks as Ubuntu VM and bare metal Ubuntu processes which is a showcase of energy efficiency. When there is no overprovisioning of cores, all the solutions consume roughly equally many CPU ticks. But when the number of tasks exceeds 36, the multiple single-core instances and the Phtread based solutions seems to introduce an overhead in terms of the need for more CPU ticks in order to consume the given workload.

In order to find out how multicore IncludeOS performs on another common processor architecture, we repeated the experiments on an AMD server. Fig. 3 illustrates the execution time of the same binary. The number of cores is 48 and there is roughly just a 10% increase in execution time when going from 1 to 48 tasks. There is no hyper-threading and hence the doubling of execution time between 48 and 96 tasks is as expected. For 30 cores and less, the Ubuntu VM performs somewhat better than multicore IncludeOS, but from then on the latter performs better. The single core IncludeOS is doing slightly better than multicore IncludeOS and the reference results of the bare metal solution is generally performing better on this platform.

As can be seen from Fig. 3, the results of the Ubuntu operating systems are quite similar when running parallel tasks as processes as when using Pthreads. For the Intel architecture, processes were most efficient.

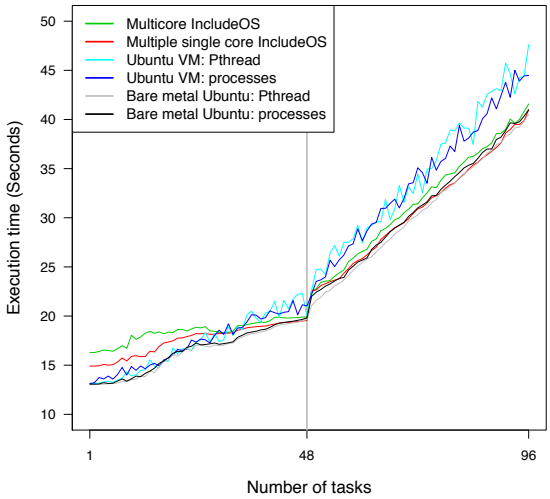


Figure 3. Execution time of prime number computation with different number of tasks in the AMD server with 48 cores.

V. Discussion

The results from the Intel and AMD servers demonstrate that multicore IncludeOS is an energy efficient operating system, which can handle large workloads efficiently compared to standard operating systems. When comparing our multicore IncludeOS solution to systems running a full-blown operating system like Ubuntu, it must be noted that the latter is much more complex and allows the programmer to utilize multicore computing in numerous ways. However, the results show the potential efficiency when developing a fully functional multicore IncludeOS kernel.

Multicore IncludeOS did, predictably, not perform as well as other solutions for small workloads that do not require many processor cores. This is due to the multicore IncludeOS design in that it boots up with only one core and as soon as it receives requests the bootstrap processor will wake up the application processors. Finally, after the cores have no workload left, they change their state to halted mode in order to save energy. With this approach, multicore IncludeOS does not use processors while there is no workload for them. The approach, however, requires a constant time for waking up cores, which means that execution time increases and this is noticeable for few and short tasks.

Apart for the case of few tasks, multicore IncludeOS performed better than the Ubuntu VM operating system on both Intel and AMD. For experiments where the number of tasks exceeded the number of physical cores, multicore IncludeOS even performed slightly better than the reference experiments running Ubuntu on a physical server on Intel. A possible reason for this behaviour could be a distinction between the multi-threading mechanisms of the KVM kernel modules and the plain Linux kernel.

For the Intel server, the multicore IncludeOS solution performed better than single core IncludeOS for most of the experiments and just as good for the rest. An additional benefit of the multicore solution is that there is no need for the management of and communication between a potentially large number of separate virtual machines.

VI. Related Work

There are today different approaches for Unikernel operating systems where some of them target specific use cases. Recent research on achieving a minimal operating system footprint have led to development of ClickOS[9], [10], Graphene[11], HermitCore[12], Drawbridge[13], HaLVM[14], OSV[15] and MirageOS[16], which are in varying levels of maturity. ClickOS aims to construct network appliances such as firewalls and loadbalancers and it does not support multiple processes. Graphene is a Linux compatible library OS, which implements a multi-process environment by creating multiple libOS instances that collaborate with each other in order to create POSIX abstraction. The HermitCore as a Unikernel operating system targets high-performance computing (HPC) and it uses multi-kernel approach for providing parallelism. Drawbridge represents Microsoft Windows library OS in which a fixed set of abstractions connect the library OS to the host kernel in order to achieve minimal footprint. HaLVM is utilizing the Glasgow Haskell Compiler toolsuite to enable

 ISBN: 978-1-61208-529-6

creating a lightweight virtual machine for the Xen hypervisor. The Haskell compiler is capable of equipping the virtual machines with multicore capabilities.

The OSV project also implemented multicore computing for its Unikernel operating system. The OSV operating system is spinlock free operating system, but it is not clear how spinlock was avoided by OSV. Multicore IncludeOS dealed with race conditions through using the atomic operations and locking bus over critical sections.

In addition, the MirageOS project wants to provide an efficient runtime for single core computing with a common immutable data store so that a large cluster of cloud-based virtual machines operate over data. In this solution, virtual machines will share data instead of logical processors. Mirage project aims to run clusters of MirageOS through multiscale compiler support in order to adopt a communication model with hardware platforms constraints [16].

As illustrated in Fig. 2a, building multiple operating system instances in a cluster requires extra time to handle workloads due to increased overhead for communication between virtual machines as well as resource consumption. Sharing the data between the virtual machines will introduce new challenges in the aspect of implementation and security. As demonstrated, a multicore operating system can efficiently achieve the same level of parallelism but with lower resource consumption. It is notable that this approach increases performance for distributed systems.

VII. Conclusion

Unikernels are designed to improve efficiency and performance but they need to utilize multicore capabilities in order to maximize performance and energy efficiency. This paper demonstrated how a multicore Unikernel approach leads to a greener cloud by adapting multicore computing to virtual environments.

The experiments demonstrated that multicore IncludeOS represents an energy efficient and cloud-optimized operating system for large workloads. Hence, it presents a real life solution as a lean and energy efficient cloud operating system with an extremely small footprint. The design principles of multicore IncludeOS improved the performance of the virtual machine as well as energy efficiency in comparison with standard operating systems and multi-kernel solutions.

A. Future Work

The multicore capability demonstrated in this paper was developed as a modular service for IncludeOS, which enable IncludeOS to easily detach it if it is not needed by the developer. Although the master/slave based structure brings flexibility to IncludeOS, it also has disadvantages such as adding extra time for waking up application processors. In a follow-up project, multicore IncludeOS will be embedded into the IncludeOS kernel by which a significant wake up time for small to medium workloads is eliminated. In addition, multicore capability can take advantage of the x2APIC standard in order to address more than 255 cores in a virtual machine, which would enable IncludeOS to utilize many-core processors in the near future. Security study of Unikernels also requires further research in order to improve reliability of Unikernel operating systems.

References

[1] M. J. Flynn and P. Hung, "Microprocessor design issues: Thoughts on the road ahead," IEEE Micro, vol. 25, no. 3, pp. 16–31, 2005. [Online]. Available: http://dx.doi.org/10.1109/MM.2005.56

[2] Xenproject. Unikernels. [Online]. Available: http://wiki.xenproject.org/wiki/Unikernels [retrieved: Jan, 2017]

[3] A. Bratterud, A. A. Walla, H. Haugerud, P. E. Engelstad, and K. Begnum, "Includeos: A minimal, resource efficient unikernel for cloud services," in 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 250–257, Nov 2015.

[4] H. Stockinger, M. Pagni, L. Cerutti, and L. Falquet, "Grid approach to embarrassingly parallel cpu-intensive bioinformatics problems," in 2006 Second IEEE International Conference on e-Science and Grid Computing (e-Science'06), pp. 58–58, Dec 2006.

[5] M. Raynal, Concurrent Programming: Algorithms, Principles, and Foundations. Springer Publishing Company, Incorporated, 2012.

[6] A. A. Kamil, "Single program, multiple data programming for hierarchical computations," Ph.D. dissertation, EECS Department, University of California, Berkeley, Aug 2012, [retrieved: Jan, 2017]. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-186.html

[7] Intel 64 and IA-32 Architectures Software Developers Manual:Basic Architecture, Intel Corporation, Sep. 2016, [retrieved: Jan, 2017]. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-1-manual.pdf

[8] Intel 64 and IA-32 Architectures Software Developers Manual: System Programming Guide, Part 1, Intel Corporation, Sep. 2016, [retrieved: Jan, 2017]. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html

[9] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling fast, dynamic network processing with clickos," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN '13, pp. 67–72. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491195

[10] J. Martins and et al., "Clickos and the art of network function virtualization," in 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp. 459–473. Seattle, WA: USENIX Association, Apr. 2014. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins

[11] C. Tsai and et al., "Cooperation and security isolation of library oses for multi-process applications," in Proceedings of the Ninth European Conference on Computer Systems, ser. EuroSys '14, pp. 9:1–9:14. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2592798.2592812

[12] S. Lankes, S. Pickartz, and J. Breitbart, "Hermitcore: A unikernel for extreme scale computing," in Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers, ser. ROSS '16, pp. 4:1–4:8. New York, NY, USA: ACM, 2016. [Online]. Available: http://doi.acm.org/10.1145/2931088.2931093

[13] D. E. Porter, G. Hunt, J. Howell, R. Olinsky, and S. Boyd-Wickizer, "Rethinking the library os from the top down," in Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). Association for Computing Machinery, Inc., March 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/rethinking-the-library-os-from-the-top-down/

[14] Galois-Inc. Halvm. [Online]. Available: https://galois.com/project/halvm/ [retrieved: Jan, 2017]

[15] A. Kivity and et al., "Osv—optimizing the operating system for virtual machines," in 2014 USENIX Annual Technical Conference (USENIX ATC 14), pp. 61–72. Philadelphia, PA: USENIX Association, Jun. 2014. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/kivity

[16] A. Madhavapeddy, R. Mortier, J. Crowcroft, and S. Hand, "Multiscale not multicore: Efficient heterogeneous cloud computing," in Proceedings of the 2010 ACM-BCS Visions of Computer Science Conference, ser. ACM-BCS '10, pp. 6:1–6:12. Swinton, UK, UK: British Computer Society, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1811182.1811191

 ISBN: 978-1-61208-529-6