Viability of Virtual Machines in HPC
A State of the Art Analysis

Jens Breitbart[1(✉)], Simon Pickartz[2], Josef Weidendorfer[1], and Antonello Monti[2]

[1] Chair for Computer Architecture, Department of Informatics, Technical University Munich, Munich, Germany
{j.breitbart,josef.weidendorfer}@tum.de
[2] Institute for Automation of Complex Power Systems, E.ON ERC, RWTH Aachen University, Aachen, Germany
{spickartz,amonti}@eonerc.rwth-aachen.de

Abstract. Virtualization is common in various areas ranging from mobiles to large data centers operated by cloud providers. Theoretically, virtualization provides various benefits that could be useful to HPC as well, e.g., suspend a large application before system maintenance or migrate a process before a node fails due to hardware malfunctioning.

In this paper, we analyze the current state of the art of virtual machines for HPC with respect to their performance and energy consumption. Furthermore, we report on our findings on the compatibility of the current HPC software stack with virtual machines and how they complicate application analysis and application tuning, as well as how current HPC hardware limits some benefits of VMs.

1 Introduction

Due to numerous advantages, virtualization is omnipresent in seemingly all areas of computing nowadays. Mobiles use virtualized instruction sets for a simplified application develop- and deployment. Embedded systems rely on virtualization to run multiple specialized operating systems for resource partitioning within one device. Desktop users employ Virtual Machines (VMs) to run different operating systems for an isolation of applications running on the same hardware, and even large commercial data centers rely on VMs for a flexible assignment of resources to their customers. As a result, most of today's processors have all kinds of hardware support for effective virtualization built-in, there is a large choice of hypervisors which can use these hardware features, and all major operating systems run well both as guests or hosts in virtualization scenarios.

Despite this pervasive support and various use cases, this technology is currently not employed in High-Performance Computing (HPC). In this paper we investigate some use cases and benefits of virtualization and perform an analysis

J. Breitbart and S. Pickartz—Supported by the Federal Ministry of Education and Research (BMBF) under Grant 01IH13004 (Project FAST).

F. Desprez et al. (Eds.): Euro-Par 2016 Workshops, LNCS 10104, pp. 721–733, 2017.
DOI: 10.1007/978-3-319-58943-5_58

of the state-of-the-art with respect to the HPC software stack, the performance, the energy consumption, and the support for current hardware. We also examine its impact on the exploitation of today's many-core architectures. The additional degree of freedom with respect to the mapping of the applications onto the hardware topology provides new facilities for performance fine-tuning. Our findings reveal that using VMs hardly results in any overhead. Furthermore, VM migration may facilitate an increased overall system throughput, but requires special support within the communication stack when using HPC interconnects. However, VMs complicate application analysis and tuning, as the new virtualization layer is not well supported in tools and various benefits of VMs like stopping and migrating are not possible with the default HPC software stack. Overall, we conclude that VMs are still not ready for large scale deployment in HPC.

The paper is organized as follows. First, Sect. 2 describes VMs and their implementation, followed by Sects. 3 and 4 which introduce the hardware and applications/benchmarks used for the rest of the paper. Section 5 shows our measurements with memory bandwidth limited and compute bound applications. Section 6 introduces possible benefits of using VMs in HPC, whereas Sect. 7 discusses if these benefits can currently be achieved. The paper finishes with an overview of related work and conclusions, in Sects. 8 and 9, respectively.

2 Virtual Machines

In this paper, we focus on system VMs, which virtualize the target hardware, i.e., a process is started on the native host hardware that itself provides a complete substitute for a system and boots a new Operating System (OS). The *hypervisor* is a software used for the management of multiple guest VMs. Modern x86 hypervisors implementations rely on hardware support such as Intel's VT-x extension [20] or AMD-V. The former was introduced with the Pentium 4 processor in 2005 and is almost identical to AMD-V, which was introduced only half a year later.

As applications running within VMs have their own virtual address space within the address space of the process representing the VM, an additional dimension of virtual memory/address translation is introduced including additional page tables (so called guest page tables). The address space of the VM is typically accessed by means of guest physical addresses and the address space of the application running within the VM by guest virtual addresses; host physical/virtual addresses refer to the native equivalents. Memory accesses from within the VM require a translation from guest virtual addresses to host physical addresses. This is performed in two steps: the guest's page tables translate guest virtual addresses to guest physical/host virtual addresses, which are again translated to host physical addresses. This so-called two level page walk is supported via VT-x in hardware by *nested paging* [2]. With VT-x the Translation Lookaside Buffer (TLB) caches mappings involve both translation levels.

VMs introduce a further degree of freedom with respect to the mapping of virtual CPUs (VCPUs) onto Hardware Thread Contexts (HTCs). Although

any VM configuration is feasible, the following three are of main interest when scheduling parallel applications on NUMA systems:

Outside pinning. Fill up the threads/processes VCPUs by VCPUs and pin theses to the physical cores in accordance to the pinning strategy suitable for the particular application. This configuration only uses as many VCPUs as needed by the application.

Outside pinning (all VCPUs). In contrast to the previous one, in this configuration all VCPUs are passed to the VM regardless of the amount of threads/processes being executed.

Host-topology. Map the host's topology onto that of the VM, i.e., perform an identity mapping of VCPUs to CPUs, and pin the threads/processes within the VM with the respective strategy from above.

3 Hardware Overview

All measurements were performed on two-socket NUMA nodes equipped with two Intel Xeon E5-2670 CPUs based on Intel's Sandy Bridge architecture. Each CPU has 8 cores resulting in a total of 16 CPU cores in the entire system whereby each core has support for two HTCs resulting in a total of 32 HTCs for the whole system. The L3 cache is shared among all cores of a CPU, both L1 and L2 cache as well as the instruction pipeline are shared among HTCs of the same core. Our systems are equipped with a total of 128 GiB of RAM (64 GiB per CPU) each. Furthermore, there are both a QDR InfiniBand Host Channel Adapter and a 1 Gbit/s Ethernet network card in the systems.

The so-called thermal design power (TDP) of each CPU in our system is 115 W, i.e., the CPU consumes about 115 W on average when all 8 cores are active and measured for a reasonably long time frame. Energy measurements were carried out by using two mechanisms: (1) the so-called Running Average Power Limit (RAPL) CPU counters which measure CPU cores, DRAM and CPU package energy consumption and (2) a MEGWARE Clustsafe, which measures the energy consumption of a whole system on the primary side.

4 Applications/Benchmarks

We used two example applications in this paper: a slightly modified version of mpiBLAST 1.6.0 and the CG solver from the numerical library LAMA [5,10]. Furthermore, we used the well known STREAM benchmark for an assessment of the impact of virtualization on memory-bound applications [14].

4.1 mpiBLAST

mpiBLAST is an application from computational biology. Using MPI-only, it is a parallel version of the original BLAST algorithm for a heuristical comparison of local similarities between genome or protein sequences from different organisms.

Due to its embarrassingly parallel nature, mpiBLAST allows for perfect scaling across tens of thousands of compute cores [11]. mpiBLAST uses a two-level master-slave approach and requires therefore at least 3 processes. The data structures used in the different steps of the BLAST search typically fit into the L1 cache resulting in a low number of cache misses. The search mostly consists of a series of indirections resolved from L1 cache hits allowing for good overlapping of different searches on the two HTCs of one core. Our modified version of mpiBLAST is available on GitHub[1]. In contrast to the original mpiBLAST we removed all `sleep()` function calls which were supposed to prevent busy waiting.

4.2 LAMA

LAMA is an open-source C++ library for numerical linear algebra. We use LAMA's standard implementation of a Conjugate Gradient (CG) solver, a hybrid OpenMP/MPI application. The library is compiled with Intel's MKL library to use basic BLAS operations within a step of the CG solver. Each solver iteration contains various global reduction operations resulting in frequent synchronization of threads as well as MPI tasks. As the involved data structures do not fit into CPU caches, the performance is fundamentally limited by main memory bandwidth and inter-core/node bandwidth for reduction operations. Thus, the CG solver obtains the best performance with just using a few cores. Consequently, it benefits from the so-called scatter pinning, i.e., threads are equally distributed among the NUMA domains and main memory bandwidth of all CPUs can be saturated with fewer threads. We use scatter pinning for all measurements involving LAMA.

5 Performance and Energy Consumption

Figure 1 presents the results of our scalability analysis of mpiBLAST and LAMA's CG solver. Figure 2 shows the average power consumption during the measurements and energy consumption of one application run. Each meter point was captured by the execution of the respective application in a loop for 30 min and averaging the individual results afterwards.

5.1 Memory Bandwidth Applications

The available memory bandwidth is slightly lower within a VM resulting an overhead of less than 5% (cf. Table 1). For the "Small" measurements[2] the TLBs can store all required translations generating hardly any page walks. Although running STREAM on the "Large" array size should result in more page walks, we notice a small overhead decrease. Hence, the effect of the additional page walk is in the order of measurement noise. As a result, memory bandwidth bound

[1] https://github.com/jbreitbart/mpifast.
[2] An array of 153 MiB (458 MiB STREAM memory consumption in total).

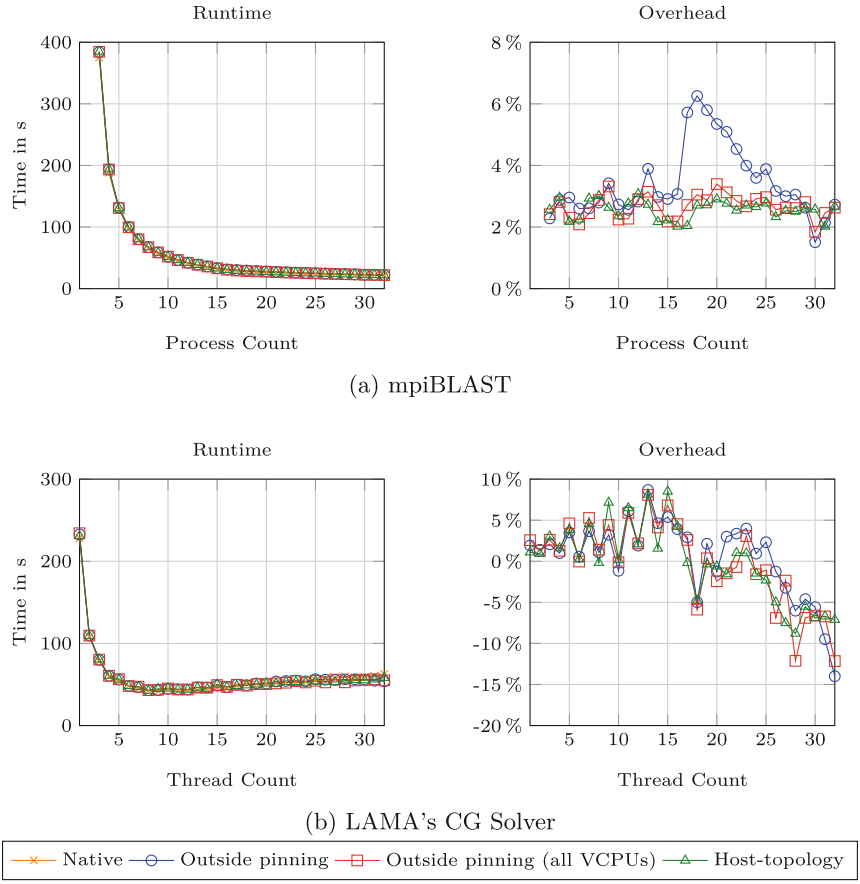
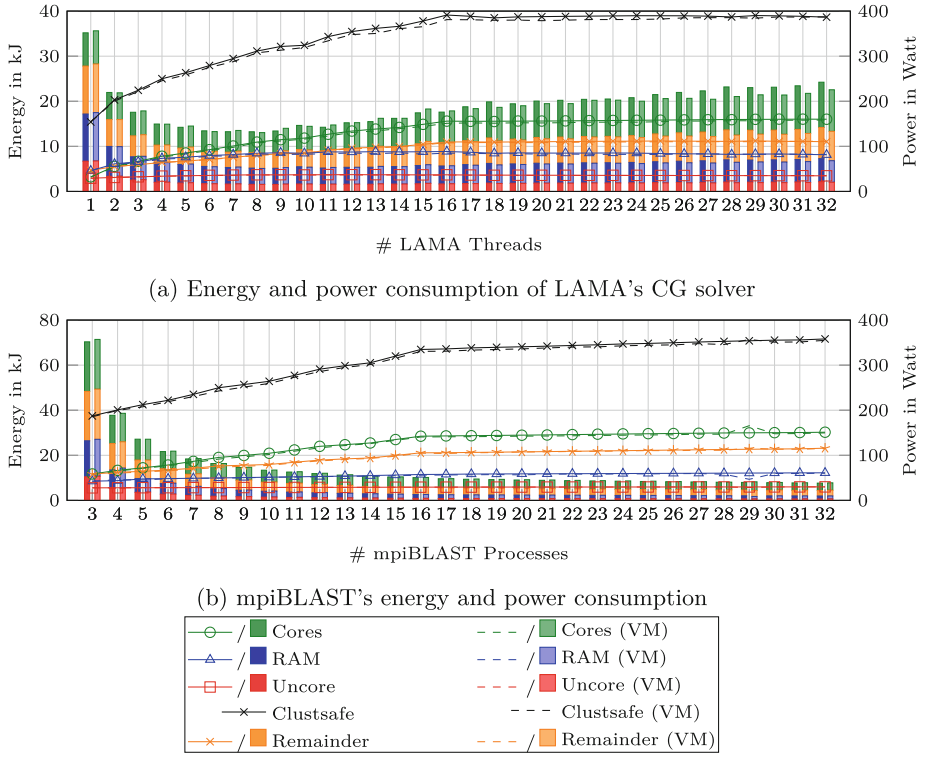


Fig. 1. Runtime of mpiBLAST and LAMA's CG Solver with varying number of processes/threads for all three VM configurations. The overhead is computed based on the native execution.

applications may suffer from a small constant performance loss, when running inside a VM. We use all 32 HTCs, compact pinning, and the host-topology configuration for the shown STREAM measurements.

As the CG solver is memory-bound, it scales up to only 11 threads with an average runtime of 41.45 s in native execution. At this meter point using a VM results in a performance penalty of around 6%. In accordance with our previous findings the CG solver only scales up to 8 threads within a VM (cf. Fig. 1) due to the lower memory bandwidth. Considering its best performance at 8 threads with 42.44 s (outside pinning) and 42.97 s (host-topology), we get an effective performance degradation of less than 4%. For thread counts exceeding 25 the VM outperforms native execution which is due less remote memory accesses within the VM. There is hardly any difference in energy consumption between native execution and using a VM (cf. Fig. 2). We expect this to be a result from different first-touch runtime behavior.



(a) Energy and power consumption of LAMA's CG solver

(b) mpiBLAST's energy and power consumption

Fig. 2. Power required while running LAMA's CG Solver/mpiBLAST (line chart) exclusively and the energy required for one run (bar chart). Uncore, RAM, and cores are measured by the RAPL counters. The remainder is the difference between the ClustSafe measurements and the sum of all RAPL counter values.

Table 1. STREAM benchmark results on array sizes of 152.6 MiB (Small) and 1.22 GiB (Large). The throughput is given in GiB/s and the Overhead (OHD) in %.

Kernel	Small			Large		
	Native	VM	OHD	Native	VM	OHD
Copy	53.23	50.76	4.64	65.22	63.69	2.34
Scale	64.27	62.79	2.30	65.99	64.73	1.91
Add	67.18	65.18	2.97	67.06	65.27	2.66
Triad	67.18	64.94	3.33	67.17	65.26	2.84

5.2 Compute Bound Applications

We do not expect compute bound applications to suffer a major performance loss when running in VM and Fig. 1 mostly confirms our expectations. The trend is the same for both, native execution and the virtualized environment showing

the best performance with 32 processes. The VM only generates little overhead of up to 3% in runtime and even less for energy consumption. However, there is one notable effect: when we start to use two HTCs per core for computation, outside pinning with a VCPU count equal to the number of processes has significant impacts on the generated overhead, e.g., about 6% for 18 processes. As mpiBLAST almost entirely works on data residing within the first-level cache, a higher number of cache miss is the most probable reason which may stem from additional system noise. However, further research is required to identify the exact cause.

6 Benefits of Virtualization

So far we have only shown that VMs have a small impact on performance and energy consumption. However, we have not discussed any of their potential benefits for HPC.

6.1 Isolation

In cloud computing VMs are often used for provisioning of isolated environments between different customers. In HPC, computing centers achieve the required isolation by a dedication of complete compute nodes to users. This strategy decreases the overall system utilization if the application is not capabable of a full system exploitation. To our knowledge, only few HPC computing centers schedule jobs at a finer granularity purely relying on Linux for a prevention of resources conflicts. Furthermore, this strategy cannot be applied if users explicitly specify thread affinities.

Nonetheless, co-scheduling jobs with diverse resource demands is known to increase overall system throughput [4]. For an evaluation of the performance penalties caused by VMs in such a scenario, we co-scheduled mpiBLAST and LAMA each within their own VM on the same host. As mpiBLAST performance is insensitive regarding the pinning, we derive its pinning from the requirements of LAMA, i.e., scatter pinning is used.

Figures 3 and 4 show the results of our measurements for both runtime and energy consumption. The efficiency is computed based on the fastest native execution, i.e., with 16 processes in co-scheduling mpiBLAST achieves about 45% of the performance of its native execution with 32 processes and LAMA achieves about 80% of the performance of its native execution with 11 threads. In general, we observe an increase of the overall application throughput similar to the results presented in [4]. The power consumption (cf. Fig. 4) is slightly increased compared to the exclusive application runs. This, however, is expected behavior as the system is doing more work while achieving a higher energy efficiency when taking the application throughput into account.

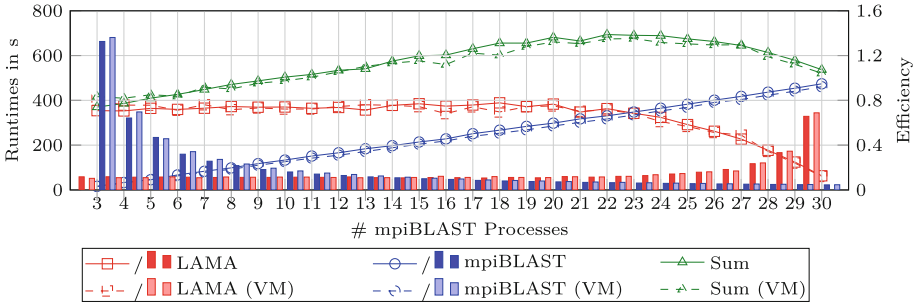


Fig. 3. Application runtime (bar chart, left y-axis) and efficiency (line chart, right y-axis) when running both LAMA and mpiBLAST concurrently. The efficiency is computed based on the most efficient exclusive application run. The x-axis shows the number of mpiBLAST processes running, the HTCs used by LAMA can be computed via 31 minus the number of mpiBLAST processes. We used 31 HTCs in total to allow for comparisons with [4].

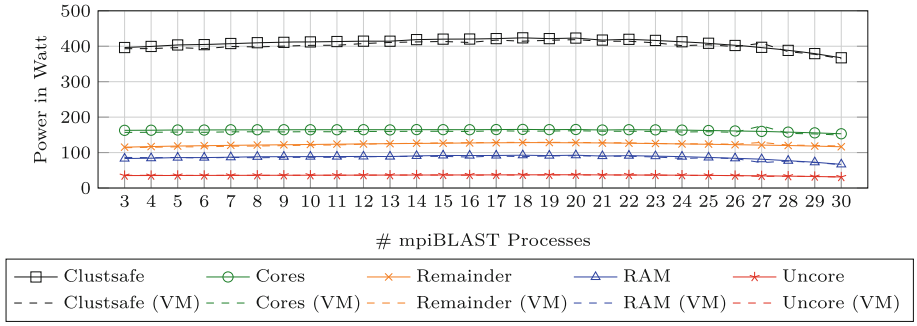


Fig. 4. Power Consumption when Co-scheduling LAMA and mpiBLAST. Uncore, RAM, and cores are measured by the RAPL counters. The remainder is the difference between the ClustSafe measurements and the sum of all RAPL counter values.

6.2 Transparent Start, Stop, and Migrate

VMs can be stopped and restarted at any time transparently from within the VM. Such a feature may ease hardware maintenances during applications runs without losing the current progress. Furthermore, VMs can also be restarted on another host or migrated from one host to another. Cloud providers leverage this feature exactly for these reasons: VMs are moved to nodes with little or no load for hardware/software maintenances or load balancing without generating application downtimes.

HPC centers typically take their system off-line for maintenance and do not apply any automatic load balancing. Resources are dedicated to jobs regardless of their actual usage. However, based on the results of our co-scheduling measurements (cf. Sect. 5), we can also show that VM migration can contribute to an increased overall system throughput. Figure 5 presents an example schedule

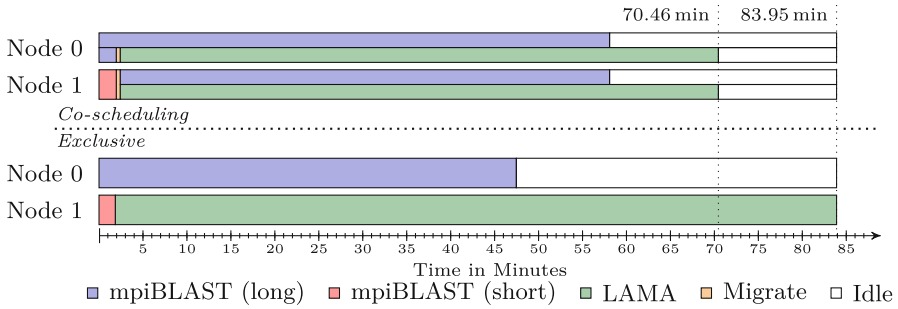


Fig. 5. Co-scheduling vs. Exclusive scheduling.

and the resulting runtime of three jobs. We expect the jobs mpiBLAST (long) and mpiBLAST (short) to be in the job queue at the beginning and LAMA to be added shortly after the start of mpiBLAST (short). The exclusive case runs natively on the hardware, whereas the co-scheduling case employs VMs in conjunction with migration reducing the runtime by about 16%. This performance increase stems from the fact that LAMA memory-bound and profits from the execution on two nodes. Executing LAMA on two nodes is only possible via migration, as Node 1 is already fully utilized by the job mpiBLAST (long). The migration time (27 s) is very short compared to the total runtime of the jobs. The energy consumption is almost identical for both scenarios with the co-scheduling case consuming 3.1 MJ and the exclusive scheduling requiring 3.2 MJ. We used Ethernet to communicate between the two nodes, as Infiniband cannot be used in conjunction with migration without changes to the software stack [16,17].

7 State of the Art

Despite the benefits of virtualization for HPC, there are certain limitations that come with this technology. This section summarizes implications thereof to the HPC software stack.

7.1 HPC Hardware/Software Support

As noted before, common hard- and software come with support for virtualization. However, major challenges are OS-bypass techniques, i.e., hardware that is directly controlled in user space such as InfiniBand or GPUs. Their employment requires support of the HPC software stack whereas two approaches for their virtualization are common: (1) the device is entirely emulated in software which typically results in unacceptable performance penalties, or (2) the guest is granted direct access to the hardware device via PCIe pass-through [1]. This is usually done in conjunction with Single Root I/O Virtualization (SR-IOV)

enabling the device virtualization in hardware such that it can be passed to multiple VMs at the same time at nearly native performance [18].

However, VM migration is impossible with attached pass-through devices. A hypervisor may unplug any device at any time, though the communication stack has to deal with such events. Most MPI implementation cannot handle an unplugging of the InfiniBand card during runtime and seemingly no hypervisor can communicate with the MPI processes to coordinate the unplug. As a result, migration is not possible for HPC applications without adjustments to the software stack [16]. NVIDIA does not provide SR-IOV support for their GPUs, but proposes a propriatary technology called GRID. As a result, GRID can only be used with a small subset of the available hypervisors (VMWare, Xen) that are directly supported by NVIDIA. AMD announced SR-IOV support at the end of 2015[3]. However, neither AMD nor NVIDIA solve the issues that come with VM migration.

Initially Kernel-based Virtual Machine (KVM) restricted VMs to run within a single NUMA domain [9] only. However, this limitation has been resolved with Version 2.0 released in 2014 allowing for the provision of virtual NUMA topologies to VMs comprising multiple VCPUs. This has implications to both GNU OpenMP and Intel OpenMP as the pinning strategies may not behave correctly, depending on the CPU/VCPU mappings.

Inter-VM intra-host communication is usually rather slow, unless a locality aware MPI implementation is deployed. Support for this feature, however, requires an experimental feature (ivshmem) that was added with the latest KVM version released in December 2015. Usually, a virtual Ethernet device (or an IB device passed through to the VM) is used for this communication path, even though shared memory would be possible [15]. Running multiple VMs with the same application on the same host is important, as the size of a VM is also the granularity at which processes may be migrated between nodes. Overall, migration is an interesting feature, however currently no HPC system scheduler leverages this mechanism for automatic load balancing. Besides being a non-trivial task by itself, VM migration limits the usable hardware.

7.2 Increased Complexity

The additional software layer that comes with virtualization adds to the complexity of the system. This layer complicates various tasks that are common in HPC. For example, we were unable to pinpoint the exact reason for the performance difference of running LAMA with a high number of threads within a VM compared to native execution. We observe a reduction of remote memory accesses when running within a VM. Further investigations with a currently unreleased NUMA simulator indicate an unstable first touch behavior, i.e., the exact distribution between NUMA nodes depends on the runtime characteristics, although it is identical in native execution among various runs. We would not

[3] http://www.amd.com/en-us/press-releases/Pages/amd-unveils-worlds-2015aug31.aspx.

be surprised if other applications reveal similar subtle (performance) bugs when running within a VM that could result in a notable performance degradation.

The virtualization layer adds a degree of freedom with respect to the affinity, i.e., the VCPUs have to be mapped to the real cores in addition to the threads/processes that map onto the VCPUs. As current OpenMP and MPI implementations lack support for this two-level pinning, we had to implement it manually for all measurements. Depending on the exact pinning strategy, we could observe efficiency variations of up to 10%.

8 Related Work

Virtualization is regularly evaluated for HPC in the recent years [12,19,21]. However, the focus is usually put on the comparison of different hypervisors on the overhead for common HPC workloads and not on the overall HPC stack. Furthermore, the migration of VMs—one of the main arguments for virtualization—is a topic of major interest [7]. Huang et al. present Nomad, a thin virtualization layer between the user processes and the InfiniBand hardware for a transparent VM migration with attached pass-through devices [8].

For effective co-scheduling, an understanding of the resource utilization and the mutual influence of applications is beneficial. Different proposals exist to come up with good predictions, often based on empirical slow-down measurements [3,6,13].

9 Conclusion

In this paper we studied the current state of the art of VMs for HPC. When looking at raw performance and energy efficiency number, VMs arguably have almost no drawback. VMs can offer various benefits for computing centers like stopping and restarting jobs or automatic load balancing, but the HPC software and hardware stack prevents the deployment of such features. The increased complexity when using VMs also adds to the burden of HPC specialists analyzing and optimizing applications. The additional indirection complicates finding the source of potential performance degradation, as we found no tool that can correlate host measurements to guest processes.

There are still various areas of research to be explored. We have preliminary OS noise/jitter measurements that indicate higher noise when using VMs, which can limit scalability of some applications. Further work is required to exactly quantify the impact. Furthermore, we plan to investigate the benefits of migration at a larger scale.

References

1. Intel Virtualization Technology for Directed I/O. Technical report, Intel Corporation (2014)
2. Bhargava, R., Serebrin, B., et al.: Accelerating two-dimensional page walks for virtualized systems. In: ASPLOS XIII: Proceedings of the 13th International Conference Architectural Support for Programming Languages and Operating System ACM (2008)
3. de Blanche, A., Lundqvist, T.: Addressing characterization methods for memory contention aware co-scheduling. J. Supercomput. **71**(4), 1451–1483 (2015)
4. Breitbart, J., Trinitis, C., et al.: Case study on co-scheduling for HPC applications. In: Proceedings of International Workshop Scheduling and Resource Management for Parallel and Distributed Systems, September 2015
5. Darling, A., Carey, L., et al.: The Design, Implementation, and Evaluation of mpiBLAST. In: Proceedings of ClusterWorld, June 2003
6. Eklov, D., Nikoleris, N., et al.: Bandwidth bandit: quantitative characterization of memory contention. In: IEEE/ACM International Symposium on Code Generation and Optimization (CGO) 2013, February 2013
7. Huang, W., Gao, Q., et al.: High performance virtual machine migration with RDMA over modern interconnects. In: 2007 IEEE International Conference on Cluster Computing (CLUSTER). IEEE (2007)
8. Huang, W., Liu, J., et al.: Nomad: migrating OS-bypass networks in virtual machines. In: VEE 2007: Proceedings of the 3rd International Conference on Virtual Execution Environments. ACM (2007)
9. Ibrahim, K.Z., Hofmeyr, S.A., et al.: Characterizing the performance of parallel applications on multi-socket virtual machines. In: CCGRID (2011)
10. Kraus, J., Förster, M., et al.: Using LAMA for efficient AMG on hybrid clusters. Comput. Sci. Res. Dev. **28**(2), 211–220 (2013)
11. Lin, H., Balaji, P., et al.: Massively parallel genomic sequence search on the Blue Gene/P architecture. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2008). IEEE (2008)
12. Luszczek, P., Meek, E., Moore, S., Terpstra, D., Weaver, V.M., Dongarra, J.: Evaluation of the HPC challenge benchmarks in virtualized environments. In: Alexander, M., et al. (eds.) Euro-Par 2011. LNCS, vol. 7156, pp. 436–445. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29740-3_49
13. Mars, J., Vachharajani, N., et al.: Contention aware execution: online contention detection and response. In: Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO) (2010)
14. McCalpin, J.D.: Memory bandwidth and machine balance in current high performance computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995. https://www.cs.virginia.edu/stream/ref.html#citing
15. Pickartz, S., Breitbart, J., et al.: Implications of process-migration in virtualized environments. In: Proceedings of 1th Workshop Co-Scheduling of HPC Applications, January 2016
16. Pickartz, S., Clauss, C., et al.: Application migration in HPC–a driver of the exascale era? In: International Conference High Performance Computing and Simulation (HPCS), July 2016
17. Pickartz, S., Clauss, C., et al.: Non-intrusive migration of MPI processes in OS-bypass networks. In: Parallel and Distributed Processing Symposium Workshops (IPDPSW) (2016)

18. Pickartz, S., Gad, R., Lankes, S., Nagel, L., Süß, T., Brinkmann, A., Krempel, S.: Migration techniques in HPC environments. In: Lopes, L., et al. (eds.) Euro-Par 2014. LNCS, vol. 8806, pp. 486–497. Springer, Cham (2014). doi:10.1007/978-3-319-14313-2_41
19. Regola, N., Ducom, J.C.: Recommendations for virtualization technologies in high performance computing. In: CloudCom (2010)
20. Uhlig, R., Neiger, G., et al.: Intel virtualization technology. Computer **38**(5), 48–56 (2005)
21. Younge, A.J., Henschel, R., et al.: Analysis of virtualization technologies for high performance computing environments. In: IEEE International Conference on Cloud Computing (CLOUD). IEEE (2011)