Application-Specific System Customization on Many-Core Platforms: The VT-ASOS Framework Position paper

Godmar Back and Dimitrios S. Nikolopoulos
Center for High-End Computing Systems
Department of Computer Science
Virginia Tech
{gback,dsn}@cs.vt.edu

Abstract

Dense shared-memory multiprocessors built with several interconnected multi-core chips, a computer organization which was recently coined as a many-core system, are emerging as a dominant architectural paradigm in high-performance computing. As many-core systems will scale to tens of cores in 2007 and hundreds of cores in the near future, effective resource allocation and scalability across the application and system software stacks become paramount. Virtualization is a technology that can potentially address this issue, as well as the issue of productivity in high-performance software development, via the provision of encapsulated and customized hardware execution environments to parallel applications. We currently explore the challenges and the opportunities of virtualization in high-end computing, in the ***VT-ASOS*** *(**V**irtualization **T**echnologies for **A**pplication-**S**pecific **O**perating **S**ystems) framework.*

1 Introduction

This paper outlines VT-ASOS, a framework for application-specific customization of the entire system software stack of multi-core systems. VT-ASOS provides components to configure a multi-core system and build an execution environment with the minimum necessary mechanisms and policies, all tailored to the application at hand. In VT-ASOS, the tasks comprising a parallel application are encapsulated in their own runtime execution environment. The execution environment consists of user-level runtime libraries, combined with a minimal, custom guest operating system that knows those application's requirements and implements only the functionality that the application needs. The primary benefit of our approach lies in harvesting the known advantages of virtualization, e.g., increased robustness and reliability through better isolation, portability, and easier software maintenance, while at the same time providing targeted support for performance-enhancing customization and optimization.

VT-ASOS uses an enhanced virtual machine monitor, built on the existing Xen VMM [1], that can optimize the allocation of such resources as CPU cores and co-processing accelerators to parallel applications, so that interference due to resource contention is minimized and applications can deploy their own customized schedulers with highly tuned parameters. This enhanced VMM supports also application-specific physical memory management policies. A performance monitoring infrastructure allows the dynamic learning of application properties, which can be subsequently communicated as resource requests to the underlying VMM.

Our preliminary results as well as those of others [5, 8] indicate that current VMM technology can support high-end computing applications with tolerable, small overhead. Figure 1 shows the execution times of 9 NAS Benchmarks, using the Class B problem size, on an Intel QX6700 quad-core Xeon processor. Execution times were collected with Linux FC6 (2.6.19 kernel) and XenLinux running on Xen 3.0 (2.6.16 kernel). We observe that Xen introduces perceptible overhead only in three benchmarks running on two or three out of the four cores. This

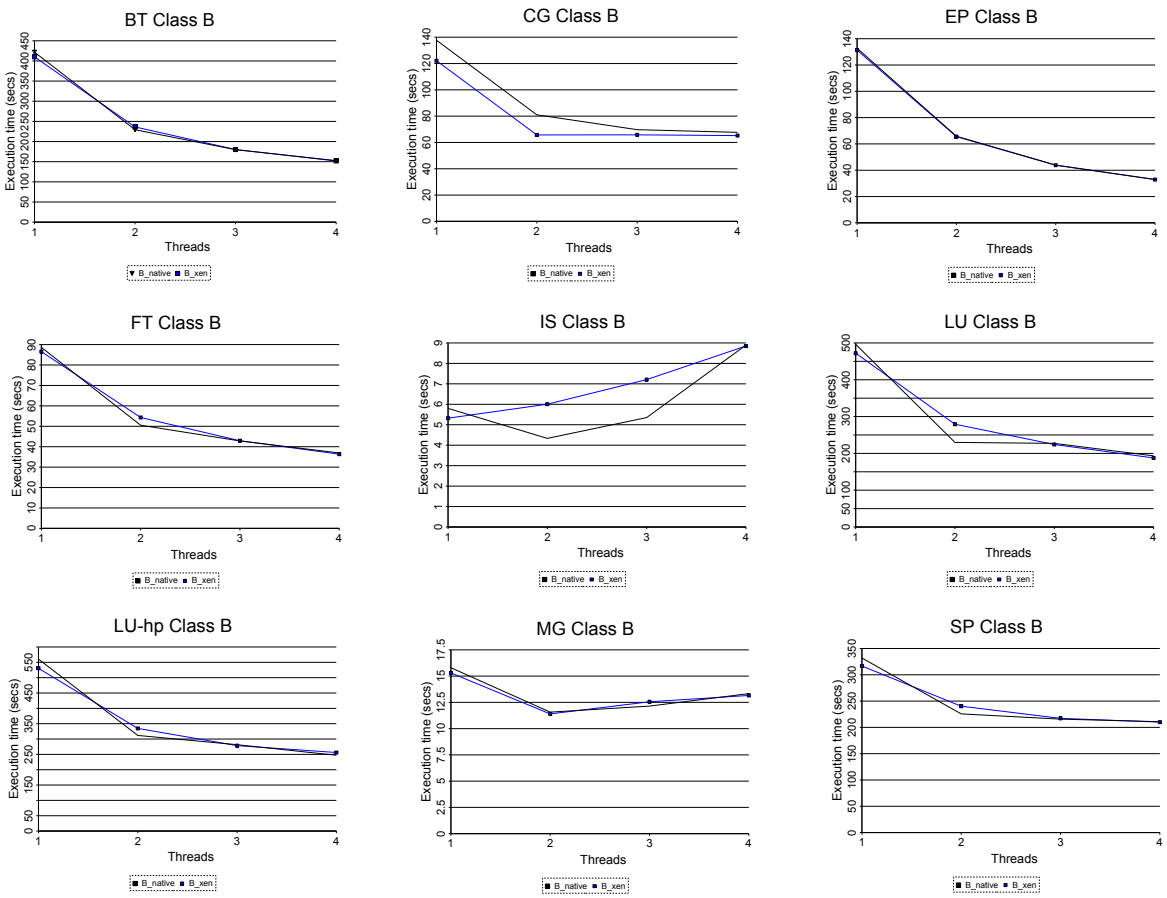

Figure 1: Execution times of NAS Benchmarks (Class B) on a quad-core Xeon QX6700 processor, using Xen 3.0 and XenLinux (kernel 2.6.16) and native Linux FC6.

overhead is likely attributed to implications of thread placement in relation to the L2 cache organization of the processor and not to inherent overhead of virtualization.

The reason for the small overhead of Xen is the use of paravirtualization. Unlike full virtualization techniques, in which the hosted guest operating systems are entirely unaware of the underlying VMM or hypervisor, paravirtualization relies on the cooperation between the guest operating system and the hypervisor. This cooperation reduces overheads because it eliminates layers of indirection. In addition to being feasible, virtualization can also be beneficial for high-end computing applications. We believe that virtualization can provide services to not only sustain, but also improve the performance of HPC applications. We discuss some application-specific system customization examples that leverages paravirtualization in the next paragraphs.

1.1 Application-specific core configurations

The introduction of multi-core execution in processors affects parallel applications in subtle and non-trivial ways. Parallel applications do not necessarily scale gracefully to any number of cores on a multi-core system (see Figure 1 for several examples among the NAS Benchmarks, such as MG, CG, IS and SP). This problem will become acute if hardware platforms move to tens of cores per chip and hundreds of cores per node in the near future. Currently, some system administrators opt to deactivate multi-core execution throughout large-scale clusters, due to undiagnosed performance anomalies in system software [1].

[1] See http://www.nersc.gov/nusers/resources/bassi/

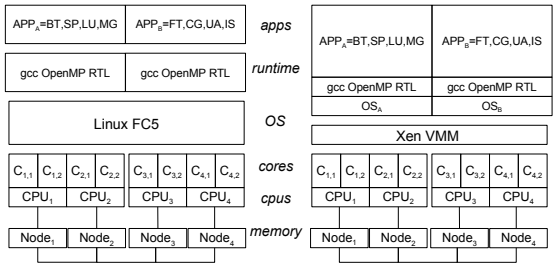

Figure 2: Workloads executed with space sharing on an Iwill blade with four dual-core AMD Socket-F processors, using Linux FC5 (left) and Xen VMM (right).

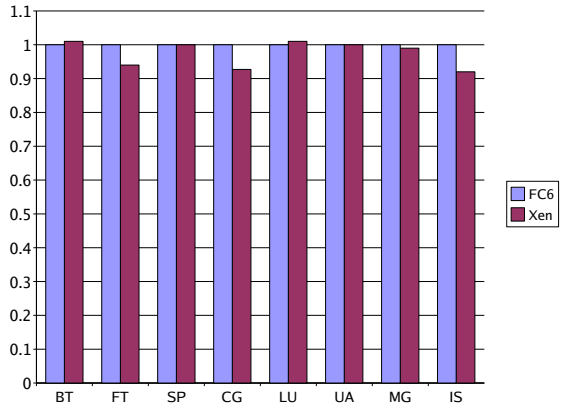

Figure 3: Normalized execution times of NAS benchmarks in multiprogram workloads with Linux FC5 and Xen.

Figure 3 illustrates the results of an experiment in which we show how paravirtualization can alleviate such a problem, by configuring cores using feedback from the application. In this experiment, we execute a multi-program workload on an Iwill AMD Opteron blade (see Figure 2) with a total of eight cores, distributed between four AMD Socket-F Opteron processors. Xen enables customized space sharing of the system, so that each partition has a core configuration which makes the best use of the available memory bandwidth for the guest OS and the application running in it. The times are obtained from four tests, each of which loads a pair of benchmarks (BT+FT, SP+CG, LU+UA, MG+IS) on the system. Each benchmark runs with four threads, so that no core is time-shared between threads. A script enforces repeated executions of each benchmark for 30 minutes. In each pair of bars, the left bar corresponds to the mean execution time of the benchmarks running in a Linux FC5 host OS (normalized to 1.0), whereas the right bar shows the mean execution times of the benchmarks running in a Linux FC5 guest OS, hosted by Xen 3.0 VMM. In the paravirtualized executions, Xen's configuration is aware of the dual-core processor memory bandwidth limitations in some of the benchmarks (FT, CG and IS) and uses a customized VM configuration, whereby each guest is isolated in two sockets and selectively uses either one core per socket or two cores per socket, depending on the application demand for memory bandwidth. Several benchmarks actually benefit from virtualization and isolation, by achieving higher performance.

In VT-ASOS, the application/runtime/guest OS entity are directly communicated to the VMM hypervisor. Rather than adjusting policies at the runtime or guest OS layer, the hypervisor itself computes an assignment of guest domains to hardware threads that optimizes resource use.

1.2 Application-specific scheduling

The schedulers in current hypervisors, such as Xen, are primarily designed for commercial server virtualization application scenarios in which different guest domains are isolated from each other and do not frequently communicate with each other. In such scenarios, achieving fairness and maximizing the utilization of a machine are paramount. Current schemes do not provide gang scheduling, or any synchronization-aware scheduling, both necessary features for parallel applications running on multi-core systems. Moreover, the interaction between the hypervisor's scheduler and the schedulers of the guest domains is not understood well, as is evidenced by the frequent turnover of often experimental scheduling algorithms in these systems. In VT-ASOS, a guest domain's scheduling requirements (such as capacity and latency requirements and synchronization patterns) are directly communicated to and enforced by the hypervisor.

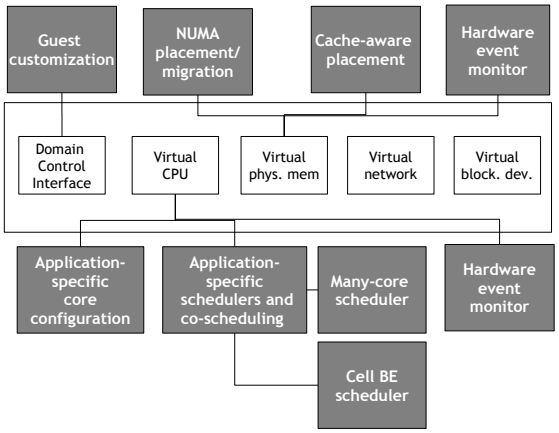

Figure 4: VT-ASOS modules.

1.3 Application-specific guest domain customizations

Multi-core HPC applications often experience performance degradation and scalability bottlenecks that are caused by other interfering OS components, in particular system programs implementing services and interrupts OS use for time keeping. By creating minimalistic guest OS that only provide the services needed by a particular application, VT-ASOS aims to eliminate these bottlenecks.

Figure 4 summarizes the primary VT-ASOS components. The rest of this position paper outlines these components in the early VT-ASOS design.

2 Application-specific processor core configurations

VT-ASOS will extend Xen to provide multi-core aware processor configuration and assignment. The current Xen provides only very limited support for placing guest domain threads onto physical cores: it enumerates all hardware (SMT) threads in depth-first fashion, and supports a "pin" operation that fixes a domain's virtual thread(s) to one or more SMT threads. Based on this assignment, it blindly attempts to keep all SMT threads busy.

VT-ASOS will extend XenLinux (the guest domain OS) such that applications can directly communicate their optimal processor allocation to it. We will extend Xen such that guest domains can directly communicate their processor requirements. Specifically, guest domains should be able to specify, the number of virtual CPUs (VCPUs) that would provide optimal concurrency, and for each VCPU, whether this VCPU would be able to coexist with others in the same core, socket, and node. In addition to such constraints, guest domains can request affinities for sets of VCPUs if they desire to place them on the same core or node.

The extended hypervisor will use the allocation requests to compute a mapping of VCPUs to physical cores; this computation can be formulated as a simple constraint-solving problem. Subsequently, this mapping will remain fixed - the hypervisor will not perform dynamic load-balancing.

Although we expect that frequently run applications will have known interference patterns, our framework will also provide a reasonable default assignment for applications for which the resource interference patterns are not known. We will include a diagnosis and monitoring framework, based on software and hardware counters, to monitor those applications' behavior. The results of the monitoring can then be used to perform a dynamic reassignment if needed.

3 Application-specific scheduling

The current Xen's scheduler is targeted at commercial server scenarios in which the hypervisor must provide fair sharing of the CPU resources for the guest domains it hosts, while being oblivious to the applications being run inside these domains and their specific requirements. Moreover, the Xen scheduler has no way to communicate with the guests's schedulers: the only information that is communicated from the guest to the hypervisor is whether a guest domain is runnable or blocked. Consequently, the hypervisor is relegated to guess work as to what its scheduling strategy should be. Xen can be configured, at boot time, to use a specific scheduler. As Xen evolved, different schedulers have been preferred in the past: borrowed-virtual time, simple EDF, and most recently a credit-based fair scheduler that bases CPU assignments on per-VPCU weights and a cap. This experimentation proves the difficulty of finding a hypervisor scheduling policy that serves all guest

domain configurations equally well.

Our approach is to tear down the abstraction wall between hypervisor and guest OS and allow the guest OS to directly communicate its preferred scheduling policy to the hypervisor. We will implement a configurable scheduler for Xen that has the ability to receive scheduling hints from the guest domains. For instance, a guest domain may request that all its virtual CPUs be subject to a gang scheduling policy, or a guest domain may request specific latency constraints for its threads.

Conversely, the hypervisor scheduler will have the ability to communicate back to the guest domain information. For instance, if the machine is heavily loaded, an application running in such a domain could then dynamically reduce the degree of parallelism it uses and turn off gang scheduling, making it easier for the underlying scheduler to meet its requirements and reducing turn-around time. In effect, this approach paravirtualizes the CPU by destroying the illusion that each virtual CPU is backed by a physical CPU, but does so for the purposes of allowing the guest domain to make more intelligent decisions.

4 Application-specific guest domain customizations

Interference from operating systems components is a major known source of performance degradation for high-performance computing applications [3, 6, 7], particularly on multi-core systems. Sources of such interference include interrupts that are used for time keeping, system daemons such as the page-out or update daemon, and other daemons such as those used to implement communication protocols such as MPI. This "noise" can adversely affect synchronicity, making it particularly difficult for bulk-synchronous applications to achieve good performance. Researchers have proposed a variety of solutions to address this problem, ranging from radical approaches such as the use of specialized, custom kernels to ad hoc approaches such as disabling daemons or keeping processors idle to absorb the noise.

We believe that the use of virtualization opens an entirely new venue to address this issue, and we propose to address both problems using guest domain customization. First, we create for every application a "right-weight" [2] system image — including libraries, guest kernel version, and system configuration parameters — that meets the needs of this application; for instance, it may or may not include virtual memory support, it may or may not include multi-threaded versions of a library, and it will only include the system programs an application needs. We envision this customization to be done at a high level, using commonly available build and runtime parameters in stock kernels, rather than using a OS library approach as in [5], because this approach is more likely to provide the type of complete application environment to which developers are accustomed.

Second, we will address the issue of noise due to timer interrupts. Others have shown that for uniprocessors, hypervisor control can reduce the impact of timer interrupt noise [4]. However, for guest OS that use multiple cores, the traditional, polling-based implementation of OS timer interrupts is more difficult because the time keeping of the different virtual CPUs must be kept in synch. Instead, we will adopt smart timers [7], an implementation of timers that does not rely on polling, but rather sets timers directly for the next event only as needed. We believe this approach to be particularly suitable in the context of a paravirtualization environment where the underlying hypervisor provides a virtual timer device for the guest domain running inside it. We will adopt the existing Linux implementation of smart timers and port it to XenLinux to test our hypothesis.

5 Conclusion

We believe that virtual machine monitors can provide the necessary customization and performance for multi-core systems software that supports parallel applications. We have outlined the preliminary design of VT-ASOS, a system that provides application-specific core configuration, scheduling, and guest domain customization.

Acknowledgement

This research is partially supported by the DOE grant DE-FG02-06ER25751.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[2] Ronald G. Minnich et al. Right-weight kernels: an off-the-shelf alternative to custom light-weight kernels. *SIGOPS Oper. Syst. Rev.*, 40(2):22–28, 2006.

[3] T. Jones et al. Improving the scalability of parallel jobs by adding parallel awareness to the operating system. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 10, Washington, DC, USA, 2003. IEEE Computer Society.

[4] Eric Van Hensbergen. The effect of virtualization on os interference. In *Workshop on Operating System Interference in High Performance Applications*. http://research.ihost.com/osihpa/osihpa-hensbergen.pdf.

[5] Eric Van Hensbergen. P.r.o.s.e.: partitioned reliable operating system environment. *SIGOPS Oper. Syst. Rev.*, 40(2):12–15, 2006.

[6] F. Petrini, D. J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 55, 2003.

[7] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick. System noise, os clock ticks, and fine-grained parallel applications. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, pages 303–312, New York, NY, USA, 2005. ACM Press.

[8] L. Youssef, R. Wolski, B. Gorda, and C. Krintz. Paravirtualization for HPC Systems. In *Proc. Workshop on Xen in High-Performance Cluster and Grid Computing*, December 2006.